Schwinn

TI-99 Tips & Tricks

Eine Fundgrube für den TI-99 Anwender

EIN DATA BECKER BUCH

Schwinn

TI-99
Tips & Tricks

Eine Fundgrube für den TI-99 Anwender

EIN DATA BECKER BUCH

Copyright (C) 1983 DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

ISBN-Nr. 3-89011-006-1

V O R W O R T

Mit fünf Büchern haben wir in der neuen Reihe der DATA BECKER BÜCHER bisher erfolgreich COMMODORE-Anwendern wertvolle Anregungen zum Umgang mit ihrem Computer und zur vollen Ausnutzung seiner Möglichkeiten gegeben.

Das sechste DATA BECKER BUCH widmen wir nun dem ebenfalls recht populären TI-00 A von TEXAS INSTRUMENTS. Obwohl er recht verbreitet ist, fand der TI-99 A bisher sehr wenig Widerhall bei Autoren und Zeitschriften.

Dem wollen wir mit dem vorliegenden Buch abhelfen. Geschrieben wurde es von Karl-Peter Schwinn, der den TI-99 wie seine Westentasche kennt.

Im kommenden Jahr wollen wir zusätzliche eine große Programmsammlung zum TI-99 herausbringen, bei der Sie, lieber Leser, mitmachen können.
Näheres hierzu finden Sie am Ende dieses Buches.

Viel Vergnügen bei der Lektüre wünscht Ihnen Ihr

Dr. Achim Becker

Düsseldorf, im Oktober 1983

Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technische Angaben und Programme in diesem Buch wurden von den Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler sind die Autoren jederzeit dankbar.

DER AUTOR ÜBER DEN AUTOR

Als ich am 9.9.1956 in Bad Dürkheim-Leistadt geboren wurde, dachte noch keine Menschenseele an Computer. Nach meiner ersten Ausbildung als Elektroinstallateur kam ich 1977 zur Bundeswehr (wo ich Heute noch bin) und lernte dort den ersten Computer meines Lebens kennen. Nach anfänglicher Skepsis wuchs meine Begeisterung für diese Maschinen derart, daß ich mich dazu entschloß, meine beruflichen Bahnen in Richtung EDV zu lenken. Den Grundstein hierfür legte ich mit einer Ausbildung zum Industriemeister der Nachrichtentechnik. Zur Zeit strebe ich den Abschluß eines staatl. gepr. Technikers der Elektrotechnik/ Fachrichtung Computertechnik an.

Karl P. Schwinn
Leistadt, im Okt. 83

Inhaltsverzeichnis

KAPITEL 1: EINFÜHRUNG IN DAS TI-BASIC..... 1

1.1 Microsoft und TI-BASIC..... 1
1.2 Vergleichsliste der Basic-Statements..... 3
1.3 TI - oder EXTENDED BASIC ?..... 8

KAPITEL 2: Umsetzen von Microsoft Programmen..... 11

2.1 Durchforsten nach fremden Befehlen..... 11
2.2 Umsetzen der Befehle..... 12
2.2.1 AND..... 13
2.2.2 OR..... 14
2.2.3 Verknüpfung von UND und ODER..... 15
2.2.4 LEFT$ (A$,Y)..... 17
2.2.5 MID$(A$,X,Y)..... 18
2.2.6 RIGHT$(A$,Y)..... 19
2.2.7 Integer Zahlen..... 21
2.2.8 Trennzeichen..... 22
2.2.9 RUN als Statement..... 23
2.2.10 Systembefehle..... 23
2.2.11 Was man sonst noch wissen sollte..... 24

KAPITEL 3: Fertige Programme zum Eintippen..... 27

3.1 Ohne Praxis geht es nicht..... 27
3.2 Ermittlung des Wochentages..... 28
3.3 Wortspielerei..... 32
3.4 Risiko..... 36
3.5 Reaktionstest..... 44
3.6 Kfz - Kostenberechnung..... 50
3.7 Hausnummer..... 56
3.8 Mathe - Trainer..... 66

Inhaltsverzeichnis

3.9 Tabellen - oder Ranglistenerstellung..... 71
3.10 Schiffe versenken..... 77
3.11 Hauptstädte..... 84
3.12 Telefonregister..... 89
3.13 Minenfeld..... 97
3.14 Textverarbeitung..... 102
3.15 Basisumwandlung..... 108

KAPITEL 4: DATENVERWALTUNG MIT DEM TI 99/4 A..... 112

4.1 Cassette oder Diskette..... 116
4.2 Cassettenoperationen..... 121
4.2.1 Speichern von Programmen..... 121
4.2.2 Einlesen von Programmen..... 122
4.2.3 Speichern von Dateien..... 122
4.2.4 Einlesen von Daten..... 126
4.3 Sequentielle Datei: Was ist das ?..... 127
4.4 Eine kleine Routine zum Schreiben auf Band..... 128
4.5 Eine kleine Leseroutine..... 129
4.6 Erweitern einer Datei..... 130
4.7 Eine universelle Datenverwaltung..... 133
4.8 Ein Programm für die Datenverwaltung..... 137

KAPITEL 5: TIP'S UND TRICK'S..... 150

5.1 Realisierung der logischen Verknüpfungsglieder.. 151
5.1.1 Die UND - Funktion..... 151
5.1.2 Die ODER - Funktion..... 152
5.1.3 Die NICHT - Funktion..... 153
5.1.4 Die Verknüpfung von UND und ODER..... 153
5.2 Die Zahl PI..... 155
5.3 Die Microsoft-Befehle LEFT$,MID$ und RIGHT$..... 156

Inhaltsverzeichnis

5.4 Ausdrucken an beliebiger Bildschirmposition..... 159
5.5 Datenspeicherung auf Cassette..... 160
5.6 Speicherplatzeinsparung bei der Daten-
verarbeitung..... 162
5.7 Eingabeüberprüfung bei menugesteuerten
Programmen..... 174
5.8 Runden von Zahlen..... 176
5.9 Wieviel Speicherplatz steht noch zur Verfügung?. 177
5.10 Programmschutz..... 178
5.10.1 Das Passwort..... 178
5.10.2 Fiktive Programmzeilen..... 180
5.11 Der PRINT USING - Befehl..... 182
5.12 Sortieren von Zahlen..... 188
5.12.1 Sortieren nach aufsteigenden Werten..... 188
5.12.2 Sortieren nach abfallenden Werten..... 189
5.13 Alphabetisches Sortieren von Strings..... 191

KAPITEL 6: GRAPHIK AUF DEM TI 99/4 A..... 195

6.1 Die Routine CALL SCREEN..... 195
6.2 Die Routine CALL COLOR..... 196
6.3 Die Routinen CALL HCHAR und CALL VCHAR..... 197
6.4 Zeichendefinition mit CALL CHAR..... 200
6.5 Die Routine CALL GCHAR..... 207
6.6 Bewegte Graphik mit dem TI 99/4 A..... 209

KAPITEL 7: DER GUTE TON IM TI..... 212

KAPITEL 8: MöGLICHKEITEN DES EXTENDED BASIC..... 217

8.1 Besonderheiten des EXTENDED BASIC..... 218
8.2 Befehle des EXTENDED BASIC..... 222

Inhaltsverzeichnis

8.2.1 ACCEPT AT..... 222
8.2.2 CALL CHARPAT..... 229
8.2.3 CALL CHARSET..... 229
8.2.4 CALL COINC..... 230
8.2.5 CALL DELSPRITE..... 230
8.2.6 DISPLAY AT..... 231
8.2.7 CALL DISTANCE..... 232
8.2.8 CALL ERR..... 233
8.2.9 IMAGE..... 233
8.2.10 CALL INIT..... 234
8.2.11 CALL LINK..... 234
8.2.12 LINPUT..... 234
8.2.13 CALL LOAD..... 235
8.2.14 CALL LOCATE..... 236
8.2.15 CALL MAGNIFY..... 236
8.2.16 MAX..... 237
8.2.17 MERGE..... 238
8.2.18 MIN..... 238
8.2.19 CALL MOTION..... 239
8.2.20 ON ERROR..... 239
8.2.21 ON WARNING..... 240
8.2.22 CALL PATTERN..... 241
8.2.23 CALL PEEK..... 242
8.2.24 PI..... 242
8.2.25 CALL POSITION..... 243
8.2.26 PRINT USING..... 243
8.2.27 REC..... 244
8.2.28 RPT$..... 244
8.2.29 CALL SAY..... 245
8.2.30 SIZE..... 245
8.2.31 CALL SPGET..... 246
8.2.32 CALL SPRITE..... 246

8.2.33 SUB..... 247
8.2.34 SUBEND..... 248
8.2.35 SUBEXIT..... 248
8.2.36 CALL VERSION..... 248
8.3 Verbesserte Befehle des TI-BASIC..... 250
8.3.1 CALL CHAR..... 250
8.3.2 CALL COLOR..... 250
8.3.3 DIM..... 250
8.3.4 EDIT..... 251
8.3.5 IF - THEN - ELSE..... 251
8.3.6 RUN..... 252

KAPITEL 9: EIN PROGRAMM ENTSTEHT..... 253

9.1 Definition der Aufgabe..... 254
9.2 Festlegen der Variablen..... 255
9.3 Formulieren der mathematischen Probleme..... 256
9.4 Festlegen der Programmstruktur..... 257
9.5 Das Flußdiagramm..... 258
9.6 Das Programm als Listing..... 259
9.7 Die Dokumentation..... 261

KAPITEL 10: SPRITES UND IHRE ANWENDUNG..... 263

10.1 Wie entstehen Sprites ?..... 263
10.2 Ein Sprite bewegt sich..... 266
10.3 Ein Sprite ändert seine Form..... 267
10.4 Ein Sprite wird größer..... 268
10.5 Sprites treffen zusammen..... 269
10.6 Wo befindet sich ein Sprite..... 271
10.7 Wie weit befinden sich zwei Sprites voneinander entfernt?..... 272

Inhaltsverzeichnis

10.8 Sprites werden gelöscht..... 273
10.9 EIn komfortabler Sprite-Editor..... 274

ANHANG..... 282

KAPITEL 1: EINFÜHRUNG IN DAS TI-BASIC

1.1 MICROSOFT und TI-BASIC

Sicherlich haben auch Sie beim Durchblättern von Fachzeitschriften feststellen müssen, daß die meisten, dort abgedruckten Programme nicht ohne weiteres in den TI 99/4A eingetippt werden können, obwohl diese Programme in BASIC geschrieben sind.

Die Ursache liegt daran, daß die Mehrzahl der anderen HOME-COMPUTER das geläufige MICROSOFT-BASIC verwenden. Der TI hat eine eigene BASIC-Version und unterscheidet sich in vielen Befehlen von den Statments des MICROSOFT-BASIC.

Warum macht nun Texas Instruments mit dem TI 99/4A eine Ausnahme ?

Diese oder ähnliche Fragen wird sich so mancher TI-Besitzer früher oder später stellen. Eine Antwort darauf könnte nur die Firma TI geben.

Sei es wie es will, man muß mit dem TI-BASIC leben und versuchen einen Weg zu finden, die reichhaltige Software in MICROSOFT-BASIC auf den TI umzusetzen.

Trotzdem muß man sagen, daß es TI gelungen ist, eine recht leistungsfähige, wenn auch eigenwillige BASIC-Version in seinen HOME-COMPUTER zu implementieren. Vor allem die sehr guten Graphikfähigkeiten und deren einfache Anwendung müssen

erwähnt werden. Der Wortschatz des TI-BASIC ist zwar um einiges schmaler als bei MICROSOFT-BASIC, man kann jedoch ohne weiteres die gleichen Probleme lösen.

Um jedem, der nicht auf die weitverbreitete Software in MICROSOFT-BASIC verzichten will Rechnung zu tragen, finden Sie in diesem Buch ein Kapitel, das sich eigens mit dem Umschreiben dieser Programme befasst.

1.2 Vergleichsliste der BASIC-Statments

Im folgenden soll durch eine kleine Liste, die keinen Anspruch auf Vollständigkeit erhebt, die Befehlssätze des TI- und MICROSOFT-BASIC gegenübergestellt werden. Vielleicht können Sie sich nach dem Durchgehen dieser Liste einen kleinen Überblick verschaffen, wie unterschiedlich diese beiden BASIC-Versionen sind.

Die Befehlssätze sind willkürlich geordnet, wobei darauf geachtet wurde, daß gleichartige oder funktionsähnliche Statmants direkt gegenüberstehen. Sollte bei einer der beiden dargestellten Versionen Funktionen oder Statments erscheinen, die es bei der anderen nicht gibt, wird das durch '***' gekennzeichnet.

TI-BASIC	MICROSOFT-BASIC
logische Operationen:	
= (gleich)	= (gleich)
<> (ungleich)	<> (ungleich)
< (kleiner als)	< (kleiner als)
> (grösser als)	> (grösser als)
<= (kleiner oder gleich)	<= (kleiner od. gl.)
>= (grösser oder gleich)	>= (grösser od. gl.)
***	NOT (logisch NICHT)
***	AND (logisch UND)
***	OR (logisch ODER)

TI-BASIC	MICROSOFT-BASIC

Systemkommandos:

LOAD 'CS1' oder 'CS2'	LOAD 'NAME'
SAVE 'CS1' oder 'CS2'	SAVE 'NAME'

(bei Befehlen, die andere periphäre Geräte als den Kassettenrekorder ansprechen, muß man sich nach der jeweiligen Bedienungsanleitung richten)

***	VERIFY (vergleichen)
RUN	RUN
***	RUN als Statment
STOP	STOP
END	END
CONTINUE oder CONT	CONTINUE oder CONT
***	POKE X,Y
***	SYS XXXXX
***	WAIT X,Y

Systemfunktionen:

***	PEEK (X)
***	USR (X)
RND	RND (X)
***	FRE (X)

Editier- und Formatkommandos:

LIST	LIST
LIST A-B	LIST A-B
LIST -B	LIST -B
LIST A-	LIST A-
REM	REM
CALL CLEAR	CLS oder CLEAR

TI-BASIC	MICROSOFT-BASIC
***	HOME
***	CRSR-Tasten
SHIFT	SHIFT
ALPHA LOCK	SHIFT LOCK

Formationsfunktionen:

TAB (X)	TAB (X)
***	SPC (X)
***	POS (X)

Stringfunktionen:

POS (A$,X$,Y)	***
LEN (X$)	LEN (X$)
STR$ (X)	STR$ (X)
VAL (X$)	VAL (X$)
CHR$ (X)	CHR$ (X)
ASC (X$)	ASC (X$)
***	LEFT$ (A$,X)
***	RIGHT$ (A$,X)
SEG$ (A$,X,Y)	MID$ (A$,X,Y)

E/A - Befehle:

INPUT A$ bzw. A	INPUT A$ bzw. A
INPUT 'Kommentar':A$	INPUT 'Kommentar';A$
CALL KEY (O,K,S)	GET A$ bzw. A
DATA A,B,C,....	DATA A,B,C,....
READ A$ bzw. A	READ A$ bzw. A
RESTORE	RESTORE
RESTORE XXX	RESTORE XXX
PRINT	PRINT

TI-BASIC	MICROSOFT-BASIC
DISPLAY	***
***	PRINT USING
CALL HCHAR	***
CALL VCHAR	***
CALL GCHAR	***
OPEN	OPEN
INPUT #X:	INPUT #X,
GET #X:	GET #X,
PRINT #X:	PRINT #X,
CLOSE #X	CLOSE X

Programmablauf:

GOTO X	GOTO X
GOSUB X	GOSUB X
IF A=X THEN Y	IF A=X THEN Y
FOR A=X TO Y	FOR A=X TO Y
STEP +/- X	STEP +/- X
NEXT	NEXT
IF A=X THEN Y ELSE Z	IF A=X THEN Y ELSE Z
RETURN	RETURN
ON X GOTO A,B,C,....	ON X GOTO A,B,C,....
ON X GOSUB A,B,C,....	ON X GOSUB A,B,C,....

Befehle zu Graphik und Ton:

CALL SOUND	Hier müssen alle Befehle
CALL COLOR	durch PEEK und POKE rea-
CALL SCREEN	lisiert werden.
CALL CHAR	

Sonderbefehle:

NUMBER oder NUM	***
BREAK XXX	***
DEF A(X)	DEF FNA(X)
RESEQUENCE oder RES	***
DIM (X,Y,Z)	DIM (X,Y,Z,...)
NEW	NEW
QUIT	***

1.3 TI- oder EXTENDED BASIC ?

Mit der Zeit und einem damit verbundenen Zuwachs an Programmiererfahrung wird sich jeder TI-Anwender mit dieser Frage beschäftigen. Wir wollen Ihnen hier eine kleine Entscheidungshilfe geben.

Diejenigen unter Ihnen, die über das, nicht gerade preiswerte Modul EXTENDED BASIC verfügen, können mir bestätigen, daß diese BASIC-Erweiterung eine wertvolle Hife zur fortgeschrittenen Programmierung mit dem TI 99/4 A ist. Will man mehr aus den selbstprogrammierten Spielen machen, um die Graphik zu verbessern, kommt man um die Anschaffung des Moduls nicht herum. Doch nicht nur im Bereich der Spielprogramme wird man effektiv unterstützt. Auch alle Problemlösungen die Sie sonst noch programmiert haben, lassen sich jetzt wesentlich erweitern bzw. verbessern. Ganz zu schweigen von der größeren Rechengeschwindigkeit (Faktor 2) die mit diesem Modul verbunden ist.

Das Modul EXTENDED BASIC erweitert Ihren TI so, daß Sie schier alle Befehle des Microsoft-Basic verfügbar haben.

Ein weiterer, großer Vorteil ist die Möglichkeit, bis zu 28 Sprites zu definieren (dies sind kleine, vom sonstigen Bildschirmgeschehen unabhängige Figuren, die ähnlich wie die Zeichen zu definieren sind). Gerade hier können Sie richtig professionelle Spiele wie z.B. SPACE INVADER oder andere Weltraum-Aktionsspiele einfach programmieren. Zum Thema Sprites gibt es in diesem Modul eine Menge Sonderbefehle, die der Steuerung der Sprites dienen. So können Sie jederzeit feststellen, ob zwei Sprites miteinander kollidieren oder wo sich im Moment ein Sprite auf dem Bildschirm befin-

det. Sogar die Struktur eines Sprites können Sie mitten im Programmlauf verändern (z.B. Explosion).

Die Graphikmöglichkeiten haben wir nun erschöpfend behandelt, so daß wir uns den anderen Vorteilen widmen können.

Hat man das Modul in den Modulschacht eingeschoben und EXTENDED BASIC gewählt, so bieten sich folgende erweiterte Möglichkeiten:

- mehrere Statments können, getrennt durch Doppelpunkte (: :) in einer Zeile programmiert werden.
- Sonderbefehle zur Erzeugung und Steuerung der Sprites sind verfügbar.
- durch die SIZE-Funktion kann man den verbleibenden Speicherplatz feststellen.
- die Zahl PI ist fest implementiert.
- der Programmablauf ist um den Faktor 2 schneller.
- durch Sonderbefehle ist es möglich bei Programmfehlern eine Fehlerbehandlung durchzuführen.
- durch DISPLAY USING oder PRINT USING kann eine formatierte Ausgabe auf dem Bildschirm erfolgen.
- mittels DISPLAY AT und INPUT AT kann man an beliebiger Bildschirmposition eine Ausgabe bzw. Eingabe vornehmen.
- Ein- und Ausgabedaten können durch Optionen überprüft werden.
- bei IF-Abfragen können nach THEN Statments folgen.

Im Kapitel EXTENDED BASIC finden Sie eine Vielzahl von Beispielen zu den oben angesprochenen Funktionen und deren Nutzung.

Alle Vorteile können hier nicht aufgeführt werden, doch der einzige Nachteil dieses Moduls soll Ihnen nicht verheimlicht werden: Das Modul EXTENDED BASIC begrenzt den Arbeitsspeicher des TI um ca. 800 BYTE ! Doch wer nimmt, bei so vielen Vorteilen, diesen kleinen Nachteil nicht gerne in Kauf ?

Alles in allem kann man jedem, vorausgesetzt er hat genügend Kleingeld, empfehlen, dieses Modul zu erwerben.

Als Nachtrag soll noch erwähnt werden, daß es möglich ist, mit dem Befehl PROTECTED ein Programm gegen Auflisten zu schützen.

Wie oben schon bemerkt, befassen wir uns im Kapitel EXTENDED BASIC noch einmal ausführlich mit den Eigenschaften und Fähigkeiten dieses Moduls.

KAPITEL 2: UMSETZEN VON MICROSOFT PROGRAMMEN

In diesem Kapitel erhalten Sie einige Hilfen und Tip's, wie man beim Umschreiben von Programmen, die in Microsoftbasic geschrieben sind, am besten vorgeht.

2.1 Durchforsten nach fremden Befehlen

Sehen Sie ein Microsoft Programm zuerst nach fremden Befehlen durch. Unter Umständen können Sie sich viel Tipparbeit ersparen, denn nicht alle Microsoftbefehle kann man ohne weiteres für den TI 99/4A umsetzen.
Gemeint sind hier Befehle wie PEEK und POKE. Ohne intime Kenntnisse des Systems, auf dem das jeweilige Programm geschrieben wurde, welches diese Befehle anwendet, kann man dieses Programmm nicht umschreiben. Zudem stehen in TI-BASIC diese Befehle nicht zur Verfügung.
Es gibt jedoch POKE oder PEEK Befehle, die für den Programmablauf nicht wesentlich sind. Um dieses festzustellen bleibt Ihnen nichts anderes übrig, als diese Programme in den TI einzugeben (alle PEEK und POKE Befehle weglassen), um zu sehen, ob das Programm läuft. Hier müssen Sie allerdings das Risiko tragen, alle Arbeit umsonst gemacht zu haben.

Wollen Sie ein solches Programm trotzdem unbedingt haben, kann man nur empfehlen, eine Liste der ROM-Adressen von diesem System zu beschaffen, um dann festzustellen, was der betreffende POKE oder PEEK Befehl eigentlich macht. Einsteigern sei von diesem Verfahren abgeraten, da die Kenntnisse hierfür (Assembler/Maschinensprache) nicht ausreichen

dürften.

Es gibt eine Vielzahl von Steuerbefehlen, die systemspezifisch sind (HOME,BEEP,MUSIC u.s.w.). Diese Befehle können Sie bei der Eingabe getrost weglassen und am Ende, wenn das Programm läuft, die Bildschirmausgabe individuell modifizieren.

2.2 Umsetzen der Befehle

An einigen Beispielen wird nun gezeigt, wie man Microsoft Befehle am besten umsetzt.
Etwas Grundsätzliches am Anfang: Beachten Sie bitte, daß der 16 K Arbeitsspeicher in TI-BASIC einem 16 K - Programm in Microsoftbasic nicht gerecht wird. Da keine Mehrfachzeilenbelegung möglich ist (außer beim EXTENDED BASIC), benötigt TI-BASIC immer etwas mehr Speicherplatz. Jede Programmzeile belegt unabhängig von den, darin enthaltenen Befehlen zunächst einmal 2 Bytes (LOW-BYTE/HI-BYTE) nur für die Zeilennummer. Schätzen Sie daher zuerst ab, ob das umzusetzende Programm in Ihren Arbeitsspeicher passt.
Besonders grossen Ärger kann auch der DIM - Befehl bereiten: Eine Überdimensionierung der benötigten Datenfelder kann während des Programmablaufs dazu führen, daß Ihr Rechner mit der Fehlermeldung 'OUT OF MEMORY' abstürtzt.

An einem kleinen Beispiel wollen wir Ihnen zeigen, wie man den Speicherplatz für eine Adressverwaltung berechnen kann:

Beispiel: Wir wollen eine Adressverwaltung aufbauen mit folgenden Eingabekriterien je Datensatz (was ein Dasatz ist erklären wir im Kapitel Datenverwaltung):

Eingabe	Zeichenzahl
NAME	15
VORNAME	12
STRASSE	15
PLZ	4
WOHNORT	10
Geasamtzeichenzahl	56
	======

Da jedes Zeichen ein Byte belegt, können wir nun sagen, daß ein Datensatz bis zu 56 BYTES belegt. Dividieren wir jetzt die Anzahl der freien Bytes (wie Sie diesen Wert ermitteln können, wird im Kapitel 'TIP'S UND TRICK'S' beschrieben) durch 56, so erhalten wir die Anzahl der Datensätze, die wir mit unserem Programm maximal speichern können. Unsere DIM-Anweisung könnte so aussehen:

```
12320 : 56 = 220
>> Die Zahl 12320 wurde willkürlich gewählt <<
DIM ADR$(220,4)
```

2.2.1 AND

Sie finden in einem Microsoft-Programm folgende Zeile:

```
100 IF B=C AND A=D THEN 200
```

Diese Zeile bewirkt eine Verzweigung nach 200, wenn B gleich C ist U N D A gleich D ist. Diese Zeile kann beim TI 99/4A so aussehen:

```
100 IF B=C THEN 101 ELSE 110
101 IF A=D THEN 200
110 .... (weitere Statments)
```

Wenn in Zeile 100 die Variable B gleich der Variablen C ist, verzweigt das Programm zur Zeile 101, um dort die zweite Bedingung zu überprüfen. Im anderen Fall fährt das Programm ab Zeile 110 fort. In Zeile 101 wird nun geprüft, ob die Variable A gleich der Variablen D ist und verzweigt, wenn die Bedingung zutrifft nach 200.

Mit dieser Methode können Sie beliebig viele UND-Bedingungen miteinander verknüpfen.

2.2.2 OR

Hier könnte eine Microsoft-Zeile so aussehen:

```
100 IF B=C OR A=D THEN 200
```

Eine Verzweigung nach 200 findet statt, wenn B gleich C O D E R A gleich D ist. Trifft keine der beiden Bedingungen zu, wird das Programm in der folgenden Zeile fortfahren.

Die Realisierung in TI-BASIC ist äußerst einfach und könnte so aussehen:

```
100 IF B=C THEN 200
101 IF A=D THEN 200
```

Diese Methode ist so leicht zu durchschauen, daß auf eine nähere Beschreibung verzichtet werden kann.

2.2.3 Verknüpfung von UND und ODER

Eine sehr leistungsfähige Programmiertechnik ist das Verknüpfen der beiden Funktionen UND und ODER. Mit Hilfe dieser Möglichkeit, kann man (fast) jedes Problem formulieren und eine Lösung herbeiführen. Das folgende Beispiel kann nur einen kleinen Einblick in die Möglichkeiten dieser Kombination geben. Mit etwas übung werden Sie sehr schnell die Vorteile, die diese Technik bietet, zu schätzen wissen.

Beispiel: Eine Verzweigung nach 200 soll dann stattfinden, wenn die Variable A gleich 15 und die Variable B gleich 8 ist. Unabhängig zu den Variablen A und B kann eine Verzweigung auch dann stattfinden, wenn nur die Variable C einen Wert von 3 angenommen hat.

Wenn Sie den Aufgabentext aufmerksam durchgelesen haben, werden Sie bereits wissen, daß zur Lösung unseres Problems eine Verknüpfung mit UND und ODER nötig ist.

In Microsoft-Basic ist das Problem sehr schnell formuliert:

```
100 IF A=15 AND B=8 OR C=3 THEN 200
```

Es gibt zwei Möglichkeiten eine Verzweigung herbeizuführen.

1. Die Variable A=15 UND die Variable B=8

   O D E R

2. Die Variable C=3

Will man dieses Problem in TI-Basic programmieren, könnte eine mögliche Lösung folgendermaßen aussehen:

```
100 IF A=15 THEN 101 ELSE 105
101 IF B=8 THEN 200
105 IF C=3 THEN 200
110 ... (weitere Statments)
```

Zuerst wird überprüft, ob der Wert der Variablen A gleich 15 ist. Wenn ja, wird die Bedingung in Zeile 101 getestet, um im JA-Falle nach 200 zu verzweigen. Trifft eine der beiden Bedingungen nicht zu, wird die Programmsteuerung zur Zeile 105 geleitet, um dort zu überprüfen ob die Variable C gleich 3 ist. Ist C=3 kann das Programm nach 200 verzweigen. Ansonsten wird das Programm in Zeile 110 fortfahren.

Gerade zu Anfang wird der Einsteiger noch Schwierigkeiten mit dieser Kombinationsmöglichkeit haben; doch mit der Zeit werden auch die schwierigsten Hürden genommen.

2.2.4 LEFT$ (A$,Y)

Diese Stringfunktion wird dazu benutzt, um aus einem String (A$) eine Anzahl (Y) von Zeichen - von links aus gesehen - abzusondern. Will man z.B. aus dem String 'TI-COMPUTER' die ersten (linken) 3 Zeichen haben, muß man mit LEFT$ so programmieren:

```
100 A$='TI-COMPUTER'
110 B$=LEFT$(A$,3)
120 PRINT B$
RUN
TI-

READY.
```

In der Zeile 110 wird der Stringvariablen B$ aus der Variablen A$ ein Teilstring von 3 Zeichen zugewiesen.

In TI-Basic kann man diese Funktion sehr einfach nachvollziehen. Da hier nur die Stringfuntion SEG$ zur Verfügung steht, müssen wir, da wir ja die ersten Zeichen eines Strings haben wollen, als Startposition die 1 eingeben:

```
100 A$='TI-COMPUTER'
110 B$=SEG$(A$,1,3)
120 PRINT B$
RUN
TI-

DONE.
```

2.2.5 MID$ (A$,X,Y)

Mit dieser Stringfunktion kann man mitten aus einem String (A$) ab einer bestimmten Position (X) eine Anzahl von Zeichen (Y) absondern. Wir wollen aus einem String, er soll diesmal 'TEXAS INSTRUMENTS' heißen, das Segment 'INST' herausfiltern. Da dieser Teilstring ab der 7. Stelle beginnt, müssen wir für X=7 eingeben. Für die Anzahl der Zeichen gibt man für Y=4 ein:

```
100 A$='TEXAS INSTRUMENTS'
110 B$=MID$(A$,7,4)
120 PRINT B$
RUN
INST

READY.
```

Die Zeile 110 weist unserer Variablen B$ wieder den Teilstring zu.

Dieses kleine Beispiel können wir fast genauso in TI-Basic programmieren. Es muß lediglich der Microsoft-Befehl MID$ durch SEG$ ersetzt werden. Hier kann man sagen, daß der Befehl MID$ die gleiche Funktion wie SEG$ hat:

```
100 A$='TEXAS INSTRUMENTS'
110 B$=SEG$(A$,7,4)
120 PRINT B$
RUN
INST

DONE.
```

2.2.6 RIGHT$ (A$,Y)

Um aus einem String (A$) eine bestimmte Anzahl (Y) von Zeichen - von rechts aus gesehen - zu bekommen, benutzt man diese Funktion. Aus unserem String 'TEXAS INSTRUMENTS' sollen die letzten 5 Zeichen abgesondert werden. Hier wird man den Befehl RIGHT$ benutzen:

```
100 A$='TEXAS INSTRUMENTS'
110 B$=RIGHT$(A$,5)
120 PRINT B$
RUN
MENTS

READY.
```

Wenn man diesen Befehl ins TI-Basic überträgt, ist schon etwas mehr Programmieraufwand nötig. Zuerst die Routine:

```
100 A$='TEXAS INSTRUMENTS'
110 A=LEN(A$)
120 IFA-4<=0 THEN 130 ELSE 140
130 B$=A$
135 GOTO 150
140 B$=SEG$(A$,A-4+1,4)
150 PRINT B$
RUN
MENTS

DONE.
```

In der Zeile 110 wird zunächst festgestellt, wie lang der String ist. Danach wird in Zeile 120 getestet, ob die Länge

des Strings ausreichend ist, um die benötigte Zeichenzahl zu bekommen. Ist die Länge nicht ausreichend, wird der Variablen B$ der komplette String aus A$ zugewiesen (130) und ausgegeben (150). In der Zeile 140 wird nun als Anfangswert der länge des Strings (A) die Zahl 4 abgezogen und 1 hinzu addiert. Der Rest ist ein Kinderspiel, wie Sie oben sehen können.

Mit den unter 2.2.4, 2.2.5 und 2.2.6 beschriebenen Stringfunktionen haben Sie bereits ein Werkzeug in der Hand, das Ihnen eine große Hilfe beim Umschreiben von Microsoft-Programmen sein wird, da diese Funktionen dort sehr häufig verwendet werden.

2.2.7 Integer Zahlen

Oftmals, beim Durchsehen von Microsoft-Programmen, werden Sie Variablenbezeichnungen entdecken, die durch das Prozentzeichen besonders gekennzeichnet sind (z.B. X%,A%,DS%). Diese Variablen enthalten Integer Zahlen.

Was sind Integer Zahlen ?

Ganze Zahlen, also Zahlen die keinen Dezimalpunkt oder keine Nachkommastellen beinhalten, nennt man Integer Zahlen. Integer Variablen verwendet man oft als Zählvariablen, z.B. in FOR .... NEXT - Schleifen oder dort, wo nur ganze Zahlen verwendet werden (z.B. Statistiken).

Beispiele für Integer Zahlen:

Integer-Zahl	Real-Zahl
500	500.023
-123	-18.5
25	0.32

Bei Ihrem TI haben Sie nur die Möglichkeit, Realzahlen (Zahlen die einen Dezimalpunkt enthalten können) durch Variablennamen zu bezeichnen. Dies ist nicht weiter tragisch, da Sie durch die INT-Funktion jederzeit Integer Zahlen erzeugen können. Da die Integervariablen zur Äußeren Kennzeichnung verwendet werden, können Sie alle derartige Variablen vernachlässigen und als Realzahlen bezeichnen.

2.2.8 Trennzeichen

Viele Basic-Versionen verwenden die unterschiedlichsten Trennzeichen (Separatoren). Trennzeichen werden verwendet, um zum Beispiel Befehle voneinander zu trennen, oder bei Ausgabe/Eingabe - Statements Formatkontrollen durchzuführen.

Im folgenden sollen die Separatoren beschrieben werden, die bei den verschiedenen Versionen durchaus üblich sind und auf Ihrem TI nicht oder anders implementiert sind.

Der Doppelpunkt (:)

In der Regel wird der Doppelpunkt als Trennzeichen zwischen verschiedenen Statements bei einer Mehrfachzeilenbelegung benutzt. In TI-Basic ist dies nicht möglich, dort müssen Sie für jeden Doppelpunkt eine neue Zeile programmieren.

Der Inverse Schrägstrich (inverse slash)

Bei einigen Basic-Versionen benutzt man anstatt des Doppelpunktes einen, von links oben nach rechts unten führenden Schrägstrich. Die Funktion ist die gleiche wie oben beschrieben.

Das Semikolon (;)

Vorwiegend bei INPUT-Befehlen eingesetzt, bewirkt dieses Semikolon das gleiche wie in TI-Basic der Doppelpunkt nach einem INPUT-Befehl.

INPUT 'NAME';A$ => TI-BASIC => INPUT 'NAME':A$

2.2.9 RUN als Statement

Die meisten BASIC-Versionen können den RUN-Befehl als Statement, also programmierbar, verwenden. Mit dem TI 99/4A hat man diese Möglichkeit nur mit dem Modul EXTENDED-BASIC. Sollten Sie einmal, beim Umsetzen eines Programmes, auf eine Zeile stoßen die RUN enthält, so ersetzen Sie diese einfach durch 'GOTO >PROGRAMMANFANG<'.
Beachten Sie, daß der RUN-Befehl immer einen RESET der belegten Speichervariablen hervorruft, d.h. alle Variablen werden mit 0 und alle Stringvariablen mit einem Leerstring ('') belegt. Vielleicht sollte man hier, am Programmbeginn, eine kleine Routine zur Initialisierung der Variablen vorsehen, um nach 'GOTO >PROGRAMMSTART<' den gleichen Effekt zu bekommen, wie er bei dem Statement RUN auftritt.

Beispiel:

```
10 REM ** INITIALISIERUNG **
20 I=0
30 A=0
40 X1=0
50 B$=''
60 TT=5
70 REM ** PROGRAMM BEGINNT HIER **
 .
 .   Programmzeilen
 .
```

2.2.10 Systembefehle

Viele Versionen verwenden bestimmte Systembefehle, um Unterprogramme in Maschinensprache aufzurufen. Diese Befehle

können entweder selbst programmierte Maschinenroutinen aufrufen oder direkt ins Betriebssystem gehen. Eine Möglichkeit zum Umsetzen in TI-Basic besteht hier nicht (ähnlich wie bei PEEK und POKE). Mit dem TI können Sie lediglich den CALL-Befehl für den Aufruf eines fest vorgegebenen Unterprogramms benutzen. In Microsoft-Programmen werden die folgenden Befehle am häufigsten verwendet:

SYS	(Aufruf eines Unterprogrammes)
USR (X)	(Aufruf eines Unterprogrammes mit Übergabeparameter)

2.2.11 Was man sonst noch wissen sollte

Zum Schluß dieses Kapitels werden wir noch einige Statements beschreiben, die auch - bis auf einen - in TI-Basic vorhanden sind, jedoch in Microsoft-Basic eine andere SYNTAX haben.

RND (X) -- Wie beim TI wird auch hier eine Zufallszahl zwischen 0 und 1 erzeugt. Es muß allerdings ein Argument mit angegeben werden (X). Dieses Argument hat bei den verschiedenen Versionen unterschiedliche Bedeutung. Einmal dient es als Blindargument, das programmiert werden muß aber nicht benutzt wird, zum andern kann es den Bereich angeben, innerhalb dessen die Zufallszahl liegen soll. Beispiel:

RND (1) => Blindargument
RND (6) => Zahl zwischen 1 und 6

GET X$ -- Diese Funktion dient zum holen eines Zeichens aus dem Tastaturpuffer. Solche Funktionen entsprechen dem CALL KEY-Befehl. Während bei dem Befehl GETX$ direkt das entsprechende Zeichen in der Variablen X$ steht, muß man beim CALL KEY-Statement, das nur den ASCII-Code des eingegebenen Zeichens liefert, mittels CHR$(K) einen String erzeugen.
Beispiel:

```
100 GETX$:IFX$=''THEN 100
110 PRINT X$
```

Hier die TI-Version:

```
100 CALL KEY (0,K,S)
110 IF S=0 THEN 100
120 X$=CHR$(K)
130 PRINT X$
```

SPC(X) -- Wenn Sie auf dieses Statement stoßen, verwechseln Sie es bitte nicht mit der TAB(X)-Funktion. Durch diesen Befehl kann man zwischen zwei Ausdrücke einen Freiraum erzeugen, während bei der TAB(X)-Funktion immer ein Bezug zum Zeilenanfang besteht. Diese Funktion ist in TI-Basic nicht vorhanden. Beispiel:

```
100 PRINT 'TEXAS';'      ';'Instruments'
RUN
TEXAS      INSTRUMENTS

DONE.
```

Mit der SPC-Funktion kann man dies so programmieren:

```
100 PRINT 'MICROSOFT';SPC(5);'BASIC'
RUN
MICROSOFT     BASIC
```

Wenn Sie dieses Kapitel aufmerksam durchgelesen und alles verstanden haben, wird das Umschreiben von fremden Programmen auf Ihren TI 99/4A ein Kinderspiel sein.

Im Kapitel Tip's und Trick's werden wir einen Teil, der eben beschriebenen Funktionen noch einmal ausführlicher behandeln.

>>>>> Wir wünschen bei der Umschreibarbeit viel Erfolg <<<<

KAPITEL 3: FERTIGE PROGRAMME ZUM EINTIPPEN

3.1 Ohne Praxis geht es nicht

Wir wollen Ihnen in unserem Buch nicht nur trockene Theorie bieten. Im folgenden haben wir sorgfältig getestete Programme, die alle lauffähig sind, abgedruckt. Sollte eines eines der Programme nach der Eingabe nicht laufen, so prüfen Sie bitte zuerst, ob sich ein Tippfehler eingeschlichen hat. Natürlich kann man nicht ausschließen, daß sich bei der Fertigstellung des Buches ein Druckfehler eingemogelt hat.

Bei der Auswahl der Programme wurde versucht, ein möglichst breites Ineressengebiet abzudecken, um allen Ansprüchen unserer Leser gerecht zu werden. Vom Spiel bis hin zur Textverarbeitung haben wir recht leistungsfähige Programme erstellt. Lassen Sie sich nicht davon abhalten, die Programme nach Herzenslust zu modifizieren, denn dadurch können Sie Ihre eigenen Programmierfähigkeiten nur verbessern.

Alle Programme sind so aufgebaut, daß Sie ohne größere Probleme die Programmiertechnik, die angewandt wurde, durchschauen können. Dies ist ein weiterer Weg, Ihnen zu zeigen, wie man Probleme programmtechnisch lösen kann.

Bleibt noch, Ihnen viel Spaß an der Tipparbeit und viel Freude an den laufenden Programmen zu wünschen!

3.2 Ermittlung des Wochentages

Dieses kleine Programm ermöglicht es Ihnen, den Wochentag eines bestimmten Datums, das Sie vorher eingeben müssen, zu berechnen. Sie können mit diesem Programm jedes Datum bis zurück zum 15.10.1582 eingeben. An diesem Datum wurde ja bekanntlich eine Zeitverschiebung vorgenommen, so daß auf den 4.10.1582 der 15.10.1582 folgte. Bei der Berechnung des Tages codiert das Programm die einzelnen Tage mit folgenden fortlaufenden Ziffern:

1=> Montag
2=> Dienstag
3=> Mittwoch
4=> Donnerstag
5=> Freitag
6=> Samstag
0=> Sonntag

Programmbeschreibung:

Zeilennummer	Erläuterungen
1 - 80	Das Programm wird initialisiert, die benötigten Formeln werden definiert und die Wochentage werden eingelesen.
90 - 97	Eingabemenu mit Überprüfung der eingegebenen Daten (97).
210 - 220	Das eingegebene Datum wird auf Gültigkeit überprüft.

Zeilennummer	Erläuterungen
225 - 280	Wenn ein falsches Datum eingegeben wurde, wird hier eine Fehlermeldung ausgegeben.
132	In dieser Zeile wird der ermittelte Wochentag auf dem Bildschirm ausgegeben.

Die restlichen Zeilen dienen der Errechnung der Tagesziffern (0 - 6), aus der der Wochentag abgeleitet wird.

Achtung: Das folgende Programm wurde für EXTENDED BASIC geschrieben. Für Umschreibarbeiten in TI-BASIC sehen Sie bitte im Kapitel EXTENDED BASIC nach, was die einzelnen Befehle für eine Bedeutung haben. Es wurden keine Befehle verwendet, die nicht auch in TI-BASIC zu realisieren wären.

```
1 REM *********************
2 REM ** SVE - SOFTWARE  **
3 REM *********************
10 REM ERMITTLUNG DES WOCHENTAGES
15 CALL CLEAR
18 DISPLAY AT (1,1):"*ERMITTLUNG DES WOCHENTAGES*"
20 DEF I(X)=SGN(X)*INT(ABS(X))
25 DEF MD(X)=INT((X/7-INT(X/7))*7+.5)
30 DIM WT$(6)
40 RESTORE
50 DATA MONTAG,DIENSTAG,MITTWOCH,DONNERSTAG ,FREITAG,SAMSTAG,SONNTAG
60 FOR V=0 TO 6
70 READ WT$(V)
80 NEXT V
90 DISPLAY AT(10,10):"TAG  :"
91 DISPLAY AT(12,10):"MONAT:"
92 DISPLAY AT(14,10):"JAHR :"
95 ACCEPT AT(10,17)BEEP VALIDATE(DIGIT)SIZE(2):TAG
96 ACCEPT AT(12,17)BEEP VALIDATE(DIGIT)SIZE(2):MN
97 ACCEPT AT(14,17)BEEP VALIDATE(DIGIT)SIZE(4):JAHR
98 IF TAG>31 OR MN>12 THEN 90
100 DU=JAHR+MN/12+TAG/365
110 GOSUB 150
120 JT=MD(JD)
125 IF JT=6 THEN 1000
130 PRINT : :
131 CALL CLEAR
132 PRINT "DER ";TAG;".";MN;".";JAHR;" IST EIN :"
133 PRINT
134 PRINT
135 PRINT TAB(10);WT$(JT)
138 PRINT : :
140 GOTO 1500
150 Y=JAHR
160 M=MN
170 IF MN<3 THEN 400
180 TE=I(365.25*Y-.75)
190 IF Y<0 THEN 500
200 JD=TE+I(30.6001*(M+1))+TAG+1720994.5
210 IF DU<1582.8443 THEN 330
220 IF DU>1582.8744 THEN 300
225 CALL CLEAR
230 PRINT CHR$(7)
240 PRINT "FALSCHE EINGABE :"
250 PRINT
260 PRINT "AUF DEN 4.10.1582"
265 PRINT
266 PRINT
270 PRINT "FOLGTE DER 15.10.1582!"
280 PRINT
282 FOR RI=1 TO 2000
283 NEXT RI
285 CALL CLEAR
290 GOTO 90
300 A=I(Y/100)
310 B=2-A+I(A/4)
320 JD=JD+B
330 JD=JD+.5
```

```
340 RETURN
400 Y=Y-1
410 M=M+12
420 GOTO 180
500 TE=I(365.25*Y-.75)
510 GOTO 200
1000 JT=-1
1010 GOTO 130
1500 FOR II=1 TO 2000
1510 NEXT II
1512 CALL CLEAR
1515 PRINT "WEITERE BERECHNUNGEN ? (J/N)"
1520 INPUT F$
1525 IF F$="J" THEN RUN
1530 END
```

3.3 Wortspielerei

Vorbild für das Programm ist die TV-Sendung 'Auf los geht's los' von Blacky Fuchsberger. Es geht wie dort darum, unbekannte Worte zu erraten.

Bei diesem unterhaltsamen Spiel können sich beliebig viele Personen beteiligen. Einer der Spieler denkt sich ein Wort aus, das er in den Computer eintippt, während die anderen Teilnehmer nicht sehen sollten, welcher Begriff eingegeben wird.

Nun wird der, zu erratende Begriff kurz und möglichst humorvoll umschrieben und der Ratespaß kann beginnen.

Wie in der TV-Show werden zuerst Buchstaben geraten die, sofern sie in dem gesuchten Begriff enthalten sind, an der richtigen Stelle dargestellt werden. Zuvor muß noch erwähnt werden, daß die Anzahl der Buchstaben (also die Länge des Begriffes) in Form von Strichen auf dem Bildschirm dargestellt werden.

Beispiel: Suchwort 'COMPUTER'

Nehmen wir an, die Rater nennen den Buchstaben 'P', dann wird die Darstellung so aussehen:

---P----

Programmbeschreibung:

Zeilennummer	Erläuterungen
10 - 170	Hier stehen die Spielregeln des Programmes.
180 - 210	In diesen Zeilen wird das Suchwort eingegeben.
220 - 250	Hier wird die Darstellung auf dem Bildschirm mit der Buchstabenabfrage erzeugt. Es ist zu beachten, daß nach Eingabe eines Buchstabens das Drücken von 'ENTER' entfällt. Wenn Sie glauben, das Wort erraten zu haben, geben Sie einfach '0' ein und Sie werden nach dem Begriff gefragt.

Alle anderen Zeilen dienen zur Berechnung der Buchstabenpositionen und der Anzeige, ob richtig oder falsch geraten wurde. In den Zeilen 480 - 650 wird eine kleine Melodie erzeugt, die bei einem Erfolg ertönen wird.

Achtung: Auch dieses Programm ist in EXTENDED BASIC geschrieben. Bitte verfahren Sie so, wie unter 3.2 beschrieben wurde.

```
1 REM ********************
2 REM ** SVE - SOFTWARE **
3 REM ********************
10 CALL CLEAR
20 DISPLAY AT(1,1):"******WORTSPIELEREI******"
30 DISPLAY AT(5,1):"FUER 'X' SPIELER!!!"
40 DISPLAY AT(7,1):"EIN SPIELER GIBT EIN WORT"
50 DISPLAY AT(9,1):"EIN, DASS VON DEN ANDEREN"
60 DISPLAY AT(11,1):"SPIELERN ERRATEN WERDEN "
70 DISPLAY AT(13,1):"MUSS."
80 DISPLAY AT(18,1):"** MAX. 15 BUCHSTABEN **"
90 DISPLAY AT(24,1):"MIT LEERTASTE WEITER"
100 CALL KEY(0,K,S):: IF S=0 THEN 100
110 DISPLAY AT(5,1)ERASE ALL:"DIE RATER GEBEN EINEN"
120 DISPLAY AT(7,1):"BUCHSTABEN EIN, DER AUF"
130 DISPLAY AT(9,1):"DEM BILDSCHIRM AN DER"
140 DISPLAY AT(11,1):"RICHTIGEN POSITION DAR-"
150 DISPLAY AT(13,1):"GESTELLT WIRD!"
160 DISPLAY AT(24,1):"MIT LEERTASTE WEITER"
170 CALL KEY(0,K,S):: IF S=0 THEN 170
180 DISPLAY AT(10,1)ERASE ALL:"WELCHES WORT:"
190 ACCEPT AT(10,14)BEEP VALIDATE(UALPHA)SIZE(15):A$
200 CALL CLEAR
210 M=LEN(A$)
220 DISPLAY AT(1,5):"WELCHER BUCHSTABE?"
230 DISPLAY AT(24,1):"WORT ERRATEN?==>'0' DRUECKEN"
240 DISPLAY AT(10,10):RPT$("_",M)
250 ACCEPT AT(1,24)BEEP VALIDATE(UALPHA,"0")SIZE(1):B$
260 IF B$="0" THEN 330
270 SC=SC+1
280 FOR Z=1 TO M
290 C$=SEG$(A$,Z,1)
300 IF C$=B$ THEN DISPLAY AT(10,9+Z)SIZE(1):B$
310 NEXT Z
320 GOTO 250
330 DISPLAY AT(1,1):"WELCHES WORT?"
340 ACCEPT AT(1,14)BEEP VALIDATE(UALPHA)SIZE(15):D$
350 IF D$<>A$ THEN DISPLAY AT(15,1):"LEIDER FALSCH GERATEN!!!!":: GOTO
440
360 CALL CLEAR
370 DISPLAY AT(5,1):"BRAVO!!RICHTIG GERATEN."
380 GOSUB 480
390 DISPLAY AT(9,1):"SIE HATTEN";SC+1;" VERSUCHE"
400 DISPLAY AT(24,1):"NOCHMAL (J/N)?"
410 ACCEPT AT(24,15)BEEP VALIDATE("JN")SIZE(1):F$
420 IF F$="N" THEN STOP
430 GOTO 180
440 FOR Z=1 TO 1000 :: NEXT Z :: DISPLAY AT(15,1):E$
450 DISPLAY AT(1,1):E$
460 DISPLAY AT(1,5):"WELCHER BUCHSTABE?"
470 GOTO 250
480 CALL SOUND(200,130,0,261,0,523,0)
490 CALL SCREEN(3)
500 CALL SOUND(200,146,0,293,0,587,0)
510 CALL SCREEN(5)
520 CALL SOUND(200,164,0,329,0,659,0)
530 CALL SCREEN(7)
540 CALL SOUND(200,174,0,349,0,698,0)
```

```
550 CALL SCREEN(11)
560 CALL SOUND(200,195,0,391,0,783,0)
570 CALL SCREEN(13)
580 CALL SOUND(200,220,0,440,0,880,0)
590 CALL SCREEN(14)
600 CALL SOUND(200,246,0,493,0,987,0)
610 CALL SCREEN(4)
620 CALL SOUND(500,261,0,523,0,1046,0)
630 CALL SCREEN(8)
640 FOR Z=1 TO 500 :: NEXT Z
650 RETURN
```

3.4 Risiko

Risiko ist ein Würfelspiel, das ich vor einiger Zeit mit guten Freunden spielte. Auf der Suche nach neuen Programmideen habe ich mich an dieses Glücksspiel zurückerinnert und es programmiert.
Auf dem Bildschirm wird eine Tabelle generiert, die aus 3 Spalten und 10 Reihen besteht. Jede Reihe hat einen bestimmten Geldwert: Die erste Reihe DM 0.20, die zweite DM 0.40 u.s.w. bis DM 2.00.
Nun würfeln zwei Spieler abwechselnd mit drei Würfeln und setzen die Summe der Augen auf eine der Reihen. Wer in einer Reihe die höchste Augenzahl erreicht, gewinnt den Geldbetrag, der für die betreffende Reihe festgelegt ist. Zum Schluß werden die Gewinnsummen beider Spieler verrechnet, um daraus die endgültige Summe zu bekommen.

Ich schlage vor, daß sich die Spieler mit Streichhölzern 'auszahlen', da man bei diesem Spiel sehr schnell arm werden kann.

Programmbeschreibung:

Zeilennummer	Erläuterungen
10 - 90	Definition der benötoigten Graphikzeichen.
100 - 160	Abfrage, ob eine Spielanleitung benötigt wird.
175 - 210	Hier geben die Spieler Ihre Namen ein.

Zeilennummer	Erläuterungen
220	Dimensionierung der benötigten Array's.
255 - 360	In diesen Zeilen wird die Tabelle erzeugt.
365 - 500	Wurf des Spielers 1 wird durchgeführt.
505 - 660	Wurf des Spielers 2.
665 - 880	Spielabrechnung (Beträge werden verrechnet.
885 -1110	Unterprogramm zur Berechnung eines Würfelwurfes.
1115-2120	Die Würfel werden dargestellt und die belegten Felder werden vor doppeltem Belegen gesperrt.
2125-2150	Wer gewinnt das Spiel ?
2155-2590	Sperren der belegten Felder.
2600-2750	Hier steht die Spielanleitung.

Dieses Spiel wurde in EXTENDED BASIC geschrieben. Eine Realisierung für TI-Basic ist relativ kompliziert aber nicht unmöglich. Das Programm ist so Umfangreich, daß bei einer Anwendung mit TI-Basic Speichplatzprobleme auftreten können.

```
1 REM ********************
2 REM ** SVE - SOFTWARE **
3 REM ********************
10 CALL CLEAR
15 REM ** ZEICHENDEFINITION
20 CALL CHAR(120,"1818181818181818")
30 CALL CHAR(121,"000000FFFF")
40 CALL CHAR(131,"0000001818")
50 CALL CHAR(132,"030300000000C0C0")
60 CALL CHAR(133,"0303001818 00C0C0")
70 CALL CHAR(134,"C3C300000000C3C3")
80 CALL CHAR(135,"C3C300181800C3C3")
90 CALL CHAR(136,"C3C300C3C300C3C3")
100 DISPLAY AT(1,8):"R I S I K O"
110 DISPLAY AT(10,1):"BRAUCHEN SIE ANWEISUNGEN?"
120 DISPLAY AT(24,1):"GEBEN SIE 'J' ODER 'N' EIN!"
130 CALL KEY(0,K,S):: IF S=0 THEN 130
140 IF K<>74 AND K<>78 THEN 130
150 IF K=78 THEN 170
160 CALL ANWEISUNG
170 CALL CLEAR
175 REM NAMEN ABFRAGE
180 DISPLAY AT(5,1):"NAME SPIELER 1:"
190 DISPLAY AT(10,1):"NAME SPIELER 2:"
200 ACCEPT AT(5,16)BEEP:NA1$
210 ACCEPT AT(10,16)BEEP:NA2$
220 DIM S1(10),S2(10),GEW(10),GEW1(10)
230 FOR I=1 TO 10 :: S1(I),S2(I),GEW(I),GEW1(I)=0 :: NEXT I
240 CALL SCREEN(11)
250 W1,W2,W3,SU,X,TR,G1,G2,FLAG1=0
255 REM TABELLE DARSTELLEN
260 DISPLAY AT(4,1)ERASE ALL:"WERT";TAB(6);NA1$;TAB(15);NA2$
;TAB(25);"GEW."
270 FOR I=6 TO 24 STEP 2
280 X=X+.2
290 IF X=1 OR X=2 THEN X$=STR$(X)&".--"
295 GOTO 310
300 X$=STR$(X)&"0"
310 DISPLAY AT(I,1):X$
320 CALL HCHAR(I-1,3,121,28)
330 NEXT I
340 FOR I=7 TO 25 STEP 9
350 CALL VCHAR(4,I,120,21)
360 NEXT I
365 REM WURF SPIELER 1
370 DISPLAY AT(1,1):NA1$;"!-DRUECKE 'ENTER'!"
380 CALL KEY(0,K,S):: IF S=0 THEN 380
390 GOSUB 890
400 FLAG=1
410 SP=8
420 DISPLAY AT(3,1):"WELCHE WETTE:"
430 ACCEPT AT(3,14)BEEP:WET
440 IF WET<>.2 AND WET<>.4 AND WET<>.6 AND WET<>.8 AND WET<>1 AND
WET<>1.2 AND WET <>1.4 AND WET<>1.6 AND WET<>1.8 AND WET<>2 THEN 420
450 GOSUB 2160
460 IF FLAG1=1 THEN 420
465 REM AUSGABEFELD LOESCHEN
470 DISPLAY AT(1,1):Q$
```

```
480 DISPLAY AT(2,1):Q$
490 DISPLAY AT(3,1):Q$
500 GOSUB 1120
505 REM WURF SPIELER 2
510 DISPLAY AT(1,1):NA2$;"!-'ENTER' DRUECKEN"
520 CALL KEY(0,K,S):: IF S=0 THEN 520
530 GOSUB 890
540 FLAG=2
550 SP=17
560 TR=TR+1
570 DISPLAY AT(3,1):"WELCHE WETTE:"
580 ACCEPT AT(3,14)BEEP:WET
590 IF WET<>.2 AND WET<>.4 AND WET<>.6 AND WET<>.8 AND WET<>1 AND
WET<>1.2 AND WET<>1.4 AND WET<>1.6 AND WET <>1.8 AND WET<>2 THEN 570
600 GOSUB 2380
610 IF FLAG1=1 THEN 570
615 REM AUSGABEFELD LOESCHEN
620 DISPLAY AT(1,1):Q$
630 DISPLAY AT(2,1):Q$
640 DISPLAY AT(3,1):Q$
650 GOSUB 1120
660 IF TR<10 THEN 370
665 REM SPIELABRECHNUNG
670 CALL SCREEN(9)
680 CALL SOUND(1000,130,0,261,0,523,0)
690 CALL SOUND(1000,174,0,349,0,698,0)
700 CALL SOUND(1500,246,0,493,0,987,0)
710 DISPLAY AT(1,1):RPT$("*",28)
720 DISPLAY AT(2,1):"*****'ENTER' DRUECKEN*******"
730 DISPLAY AT(3,1):RPT$("*",28)
740 CALL KEY(0,K,S):: IF S=0 THEN 740
750 CALL CLEAR
760 IF G1<0 THEN DISPLAY AT(5,1):NA1$;" MUSS DM";ABS(G1);"ZAHLEN."
770 IF G1=0 THEN DISPLAY AT(5,1):NA1$;" GEWINNT NICHTS."
780 IF G1>0 THEN DISPLAY AT(5,1):NA1$;" GEWINNT DM";G1
790 IF G2<0 THEN DISPLAY AT(10,1):NA2$;" MUSS DM";ABS(G2);"ZAHLEN."
800 IF G2=0 THEN DISPLAY AT(10,1):NA2$;" GEWINNT NICHTS."
810 IF G2>0 THEN DISPLAY AT(10,1):NA2$;" GEWINNT DM";G2
820 IF G1>G2 THEN DISPLAY AT(15,1):"** GLUECKWUNSCH ";NA1$;" **"
830 IF G1<G2 THEN DISPLAY AT(15,1):"** GLUECKWUNSCH ";NA2$;" **"
840 DISPLAY AT(24,1):"NOCHMAL? (J ODER N)"
850 CALL KEY(0,K,S):: IF S=0 THEN 850
860 IF K<>74 AND K<>78 THEN 850
870 IF K=78 THEN END
880 CALL CLEAR :: GOTO 230
885 REM UNTERPROGRAMM WURF
890 RANDOMIZE
900 W1=INT(6*RND)+1
910 W2=INT(6*RND)+1
920 W3=INT(6*RND)+1
930 SU=W1+W2+W3
940 DISPLAY AT(2,1):"GESAMT:";SU;"--"
950 WU=W1
960 PLZ=20
970 GOSUB 1050
980 WU=W2
990 PLZ=24
1000 GOSUB 1050
1010 WU=W3
1020 PLZ=28
```

```
1030 GOSUB 1050
1040 RETURN
1050 IF WU=1 THEN CALL HCHAR(2,PLZ+2,131)
1060 IF WU=2 THEN CALL HCHAR(2,PLZ+2,132)
1070 IF WU=3 THEN CALL HCHAR(2,PLZ+2,133)
1080 IF WU=4 THEN CALL HCHAR(2,PLZ+2,134)
1090 IF WU=5 THEN CALL HCHAR(2,PLZ+2,135)
1100 IF WU=6 THEN CALL HCHAR(2,PLZ+2,136)
1110 RETURN
1115 REM DARSTELLUNG WUERFELSUMME
1120 IF WET=.2 THEN DISPLAY AT(6,SP)SIZE(3):SU ELSE 1220
1130 IF FLAG=2 THEN 1180
1140 S1(1)=SU
1150 IF S2(1)=0 THEN 2120
1160 IF S1(1)=S2(1) THEN DISPLAY AT(6,26):"2" :: G2=G2+WET :: G1=G1-WET
:: GOTO 2120
1170 I=1 :: J=6 :: GOSUB 2130 :: GOTO 2120
1180 S2(1)=SU
1190 IF S1(1)=0 THEN 2120
1200 IF S2(1)=S1(1) THEN DISPLAY AT(6,26):"1" :: G1=G1+WET :: G2=G2-WET
1210 I=1 :: J=6 :: GOSUB 2130 :: GOTO 2120
1220 IF WET=.4 THEN DISPLAY AT(8,SP)SIZE(3):SU ELSE 1320
1230 IF FLAG=2 THEN 1280
1240 S1(2)=SU
1250 IF S2(2)=0 THEN 2120
1260 IF S1(2)=S2(2) THEN DISPLAY AT(8,26):"2" :: G2=G2+WET :: G1=G1-WET
:: GOTO 2120
1270 J=8 :: I=2 :: GOSUB 2130 :: GOTO 2120
1280 S2(2)=SU
1290 IF S1(2)=0 THEN 2120
1300 IF S2(2)=S1(2) THEN DISPLAY AT(8,26):"1" :: G1=G1+WET :: G2=G2-WET
:: GOTO 2120
1310 J=8 :: I=2 :: GOSUB 2130 :: GOTO 2120
1320 IF WET=.6 THEN DISPLAY AT(10,SP)SIZE(3):SU ELSE 1420
1330 IF FLAG=2 THEN 1380
1340 S1(3)=SU
1350 IF S2(3)=0 THEN 2120
1360 IF S1(3)=S2(3) THEN DISPLAY AT(10,26):"2" :: G2=G2+WET ::
G1=G1-WET :: GOTO 2120
1370 J=10 :: I=3 :: GOSUB 2130 :: GOTO 2120
1380 S2(3)=SU
1390 IF S1(3)=0 THEN 2120
1400 IF S2(3)=S1(3) THEN DISPLAY AT(10,26):"1" :: G1=G1+WET ::
G2=G2-WET :: GOTO 2120
1410 J=10 :: I=3 :: GOSUB 2130 :: GOTO 2120
1420 IF WET=.8 THEN DISPLAY AT(12,SP)SIZE(3):SU ELSE 1520
1430 IF FLAG=2 THEN 1480
1440 S1(4)=SU
1450 IF S2(4)=0 THEN 2120
1460 IF S1(4)=S2(4) THEN DISPLAY AT(12,26):"2" :: G2=G2+WET ::
G1=G1-WET :: GOTO 2120
1470 J=12 :: I=4 :: GOSUB 2130 :: GOTO 2120
1480 S2(4)=SU
1490 IF S1(4)=0 THEN 2120
1500 IF S2(4)=S1(4) THEN DISPLAY AT(12,26):"1" :: G1=G1+WET ::
G2=G2-WET ::GOTO 2120
1510 J=12 :: I=4 :: GOSUB 2130 :: GOTO 2120
1520 IF WET=1 THEN DISPLAY AT(14,SP)SIZE(3):SU ELSE 1620
1530 IF FLAG=2 THEN 1580
1540 S1(5)=SU
```

```
1550 IF S2(5)=0 THEN 2120
1560 IF S1(5)=S2(5) THEN DISPLAY AT(14,26):"2" :: G2=G2+WET ::
G1=G1-WET :: GOTO 2120
1570 J=14 :: I=5 :: GOSUB 2130 :: GOTO 2120
1580 S2(5)=SU
1590 IF S1(5)=0 THEN 2120
1600 IF S2(5)=S1(5) THEN DISPLAY AT(14,26):"1" :: G1=G1+WET ::
G2=G2-WET :: GOTO 2120
1610 J=14 :: I=5 :: GOSUB 2130 :: GOTO 2120
1620 IF WET=1.2 THEN DISPLAY AT(16,SP)SIZE(3):SU ELSE 1720
1630 IF FLAG=2 THEN 1680
1640 S1(6)=SU
1650 IF S2(6)=0 THEN 2120
1660 IF S1(6)=S2(6) THEN DISPLAY AT(16,26):"2" :: G2=G2+WET ::
G1=G1-WET :: GOTO 2120
1670 J=16 :: I=6 :: GOSUB 2130 :: GOTO 2120
1680 S2(6)=SU
1690 IF S1(6)=0 THEN 2120
1700 IF S2(6)=S1(6) THEN DISPLAY AT(16,26):"1" :: G1=G1+WET ::
G2=G2-WET :: GOTO 2120
1710 J=16 :: I=6 :: GOSUB 2130 :: GOTO 2120
1720 IF WET=1.4 THEN DISPLAY AT(18,SP)SIZE(3):SU ELSE 1820
1730 IF FLAG=2 THEN 1780
1740 S1(7)=SU
1750 IF S2(7)=0 THEN 2120
1760 IF S1(7)=S2(7) THEN DISPLAY AT(18,26):"2" :: G2=G2+WET ::
G1=G1-WET :: GOTO 2120
1770 J=18 :: I=7 :: GOSUB 2130 :: GOTO 2120
1780 S2(7)=SU
1790 IF S1(7)=0 THEN 2120
1800 IF S2(7)=S1(7) THEN DISPLAY AT(18,26):"1" :: G1=G1+WET ::
G2=G2-WET :: GOTO 2120
1810 J=18 :: I=7 :: GOSUB 2130 :: GOTO 2120
1820 IF WET=1.6 THEN DISPLAY AT(20,SP)SIZE(3):SU ELSE 1920
1830 IF FLAG=2 THEN 1880
1840 S1(8)=SU
1850 IF S2(8)=0 THEN 2120
1860 IF S1(8)=S2(8) THEN DISPLAY AT(20,26):"2" :: G2=G2+WET ::
G1=G1-WET :: GOTO 2120
1870 J=20 :: I=8 :: GOSUB 2130 :: GOTO 2120
1880 S2(8)=SU
1890 IF S1(8)=0 THEN 2120
1900 IF S2(8)=S1(8) THEN DISPLAY AT(20,26):"1" :: G1=G1+WET ::
G2=G2-WET :: GOTO 2120
1910 J=20 :: I=8 :: GOSUB 2130 :: GOTO 2120
1920 IF WET=1.8 THEN DISPLAY AT(22,SP)SIZE(3):SU ELSE 2020
1930 IF FLAG=2 THEN 1980
1940 S1(9)=SU
1950 IF S2(9)=0 THEN 2120
1960 IF S1(9)=S2(9) THEN DISPLAY AT(22,26):"2" :: G2=G2+WET ::
G1=G1-WET :: GOTO 2120
1970 J=22 :: I=9 :: GOSUB 2130 :: GOTO 2120
1980 S2(9)=SU
1990 IF S1(9)=0 THEN 2120
2000 IF S2(9)=S1(9) THEN DISPLAY AT(22,26):"1" :: G1=G1+WET ::
G2=G2-WET :: GOTO 2120
2010 J=22 :: I=9 :: GOSUB 2130 :: GOTO 2120
2020 IF WET=2 THEN DISPLAY AT(24,SP)SIZE(3):SU ELSE 2120
2030 IF FLAG=2 THEN 2080
2040 S1(10)=SU
```

```
2050 IF S2(10)=0 THEN 2120
2060 IF S1(10)=S2(10) THEN DISPLAY AT(24,26):"2" :: G2=G2+WET ::
G1=G1-WET
2070 J=24 :: I=10 :: GOSUB 2130 :: GOTO 2120
2080 S2(10)=SU
2090 IF S1(10)=0 THEN 2120
2100 IF S2(10)=S1(10) THEN DISPLAY AT (24,26):"1" :: G1=G1+WET ::
G2=G2-WET :: GOTO 2120
2110 J=24 :: I=10 :: GOSUB 2130
2120 RETURN
2125 REM WER GEWINNT?
2130 IF S1(I)<S2(I) THEN DISPLAY AT(J,26):"2" :: G2=G2+WET :: G1=G1-WET
2140 IF S1(I)>S2(I) THEN DISPLAY AT(J,26):"1" :: G1=G1+WET :: G2=G2-WET
2150 RETURN
2155 REM KEIN FELD DOPPELT BELEGEN!
2160 IF WET=.2 AND GEW(1)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2370
2170 IF WET=.4 AND GEW(2)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2370
2180 IF WET=.6 AND GEW(3)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2370
2190 IF WET=.8 AND GEW(4)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2370
2200 IF WET=1 AND GEW(5)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2370
2210 IF WET=1.2 AND GEW(6)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2370
2220 IF WET=1.4 AND GEW(7)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2370
2230 IF WET=1.6 AND GEW(8)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO2370
2240 IF WET=1.8 AND GEW(9)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2370
2250 IF WET=2 AND GEW(10)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2370
2260 FLAG1=0
2270 IF WET=.2 THEN GEW(1)=1
2280 IF WET=.4 THEN GEW(2)=1
2290 IF WET=.6 THEN GEW(3)=1
2300 IF WET=.8 THEN GEW(4)=1
2310 IF WET=1 THEN GEW(5)=1
2320 IF WET=1.2 THEN GEW(6)=1
2330 IF WET=1.4 THEN GEW(7)=1
2340 IF WET=1.6 THEN GEW(8)=1
2350 IF WET=1.8 THEN GEW(9)=1
2360 IF WET=2 THEN GEW(10)=1
2370 RETURN
2380 IF WET=.2 AND GEW1(1)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2590
2390 IF WET=.4 AND GEW1(2)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2590
2400 IF WET=.6 AND GEW1(3)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2590
2410 IF WET=.8 AND GEW1(4)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2590
2420 IF WET=1 AND GEW1(5)=1 THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2590
2430 IF WET=1.2 AND GEW1(6)=1 THEN DISPLAY AT(1,1):"FELD SCHON
BESETZT!" :: FLAG1=1 :: GOTO 2590
2440 IF WET=1.4 AND GEW1(7)=1 THEN DISPLAY AT(1,1):"FELD SCHON
```

```
BESETZT!" :: FLAG1=1 :: GOTO 2590
2450 IF WET=1.6 AND GEW1(8)=1 THEN DISPLAY AT(1,1):"FELD SCHON
BESETZT!" :: FLAG1=1 :: GOTO 2590
2460 IF WET=1.8 AND GEW1(9)=1 THEN DISPLAY AT(1,1):"FELD SCHON
BESETZT!" :: FLAG1=1 :: GOTO 2590
2470 IF WET=2 AND GEW1(10) THEN DISPLAY AT(1,1):"FELD SCHON BESETZT!"
:: FLAG1=1 :: GOTO 2590
2480 FLAG1=0
2490 IF WET=.2 THEN GEW1(1)=1
2500 IF WET=.4 THEN GEW1(2)=1
2510 IF WET=.6 THEN GEW1(3)=1
2520 IF WET=.8 THEN GEW1(4)=1
2530 IF WET=1 THEN GEW1(5)=1
2540 IF WET=1.2 THEN GEW1(6)=1
2550 IF WET=1.4 THEN GEW1(7)=1
2560 IF WET=1.6 THEN GEW1(8)=1
2570 IF WET=1.8 THEN GEW1(9)=1
2580 IF WET=2 THEN GEW1(10)=1
2590 RETURN
2600 SUB ANWEISUNG
2610 CALL CLEAR
2620 PRINT RPT$("*",28)
2630 PRINT "******** R I S I K O *******"
2640 PRINT RPT$("*",28)
2650 PRINT :: PRINT :: PRINT
2660 PRINT "ZWEI SPIELER WUERFELN AB-   WECHSELND MIT DREI WUERFELN."
2670 PRINT "DANACH WIRD DIE SUMME DER   WUERFEL ERRECHNET UND JEDER"
2680 PRINT "SPIELER ENTSCHEIDET, WELCHEN GELDBETRAG ER DAMIT WETTEN"
2690 PRINT "WILL. WER IN EINER REIHE DIE MEISTEN PUNKTE HAT, GEWINNT"
2700 PRINT "DEN GELDBETRAG, DER IN SPALTE 1 ANGEGEBEN IST. BEI
GLEICHER"
2710 PRINT "PUNKTEZAHL GEWINNT DER, DER ZUERST GESETZT HAT."
2720 PRINT :: PRINT :: PRINT
2730 PRINT "VIEL SPASS !!!!"
2740 CALL KEY(0,K,S) :: IF S=0 THEN 2740
2750 SUBEND
```

3.5 Reaktionstest

Wie gut Ihr Reaktionsvermögen im Vergleich zu einer anderen Person ist, können Sie durch dieses kleine Programm testen.

Das Programm generiert drei verschiedene Graphikzeichen, von denen zwei den beiden Spielern zugewiesen werden. Das dritte Zeichen gilt für beide gemeinsam. Diese Zeichen werden nun an verschiedenen Bildschirmpositionen einzeln dargestellt, worauf der jeweilige Spieler, dem das betreffende Zeichen zugeteilt wurde, möglichst schnell auf seine Taste drücken muß ('M' oder 'Z').

Zu beachten ist, daß die Zeichen nur für etwa 0.5 Sekunden auf dem Bildschirm erscheinen. Innerhalb dieser Zeit muß reagiert werden. Nun kann es vorkommen, daß man unter Zeitdruck auf das Zeichen des Gegenspielers reagiert, was den Abzug von 2 Punkten zur Folge hat. Reagiert ein Spieler richtig, bekommt er einen Punkt.
Wer zuerst 15 Punkte hat, ist Sieger.

Programmbeschreibung:

Zeilennummer	Erläuterungen
10 - 25	Erzeugung der benötigten Graphikzeichen.
30 - 55	Abfrage, ob eine Spielanleitung gewünscht wird.

Zeilennummer	Erläuterungen
60 - 500	Spielanleitung und Eingabe der Spielernamen.
505 - 565	Erzeugen der Bildschirmgraphik.
570 -3075	Steuerung der einzelnen Zeichendarstellungen und Überwachung des Spielverlaufes.
4000-4540	Erkennung, welcher Spieler das Spiel gewonnen hat.

Nachtrag: Wenn beide Spieler in einer Spalte die gleiche genzahl gewettet haben, so gewinnt der Spieler, der als erster gesetzt hat.

```
10 CALL CLEAR
15 CALL CHAR(128,"3C4281818181423C")
20 CALL CHAR(129,"3C7EFFFFFFFF7E3C")
25 CALL CHAR(130,"55AA55AA55AA55AA55AA")
30 PRINT "WOLLEN SIE INSTRUKTIONEN?"
35 PRINT
40 PRINT TAB(7);"(J/N)"
45 CALL KEY(0,K,S)
50 IF S=0 THEN 45
55 IF K=ASC("N") THEN 460
60 CALL CLEAR
65 PRINT "*******REAKTIONSTEST*******"
70 PRINT
75 PRINT
80 PRINT "DIESES SPIEL WIRD VON"
85 PRINT
90 PRINT "ZWEI PERSONEN GESPIELT."
95 PRINT
100 PRINT "AN VERSCHIEDENEN POSITIONEN"
105 PRINT
110 PRINT "DES BILDSCHIRMS WERDEN"
115 PRINT
120 PRINT "VERSCHIEDENE ZEICHEN DAR-"
125 PRINT
130 PRINT "GESTELLT.DIESE ZEICHEN"
135 PRINT
140 PRINT "WERDEN FUER CA. 0,5 SEK."
145 PRINT
150 PRINT "DARGESTELLT."
155 PRINT
160 PRINT
165 FOR Z=1 TO 2000
170 NEXT Z
175 CALL CLEAR
180 A1$="LINKER SPIELER"
185 A2$="RECHTER SPIELER"
190 A3$="BEIDE SPIELER"
195 PRINT "JEDER SPIELER BEKOMMT"
200 PRINT
205 PRINT "EIN ZEICHEN ZUGEWIESEN,"
210 PRINT
215 PRINT "DESSEN ERSCHEINEN ER MIT"
220 PRINT
225 PRINT "SEINER TASTE ('M' ODER 'Z')"
230 PRINT
235 PRINT "QUITTIEREN MUSS.KOMMT DAS"
240 PRINT
245 PRINT "ZEICHEN FUER BEIDE,GEWINNT"
250 PRINT
255 PRINT "DER,DER ZUERST DRUECKT."
260 PRINT
265 "BEI FALSCHEM DRUECKEN WERDEN"
270 PRINT
275 PRINT "2 PUNKTE ABGEZOGEN."
280 PRINT
285 PRINT "HIER DIE ZEICHEN:"
290 FOR Z=1 TO 2000
295 NEXT Z
```

```
300 CALL CLEAR
305 CALL HCHAR(5,6,129)
310 CALL HCHAR(11,6,128)
315 CALL HCHAR(17,6,130)
320 A1=LEN(A1$)
325 A2=LEN(A2$)
330 FOR Z=1 TO A1
335 A4$=SEG$(A1$,Z,1)
340 A4=ASC(A4$)
345 CALL HCHAR(5,10+Z,A4)
350 NEXT Z
355 FOR Z=1 TO A2
360 A5$=SEG$(A2$,Z,1)
365 A5=ASC(A5$)
370 CALL HCHAR(11,10+Z,A5)
375 NEXT Z
380 A3=LEN(A3$)
385 FOR Z=1 TO A3
390 A6$=SEG$(A3$,Z,1)
395 A6=ASC(A6$)
400 CALL HCHAR(17,10+Z,A6)
405 NEXT Z
410 FOR Z=5 TO 17 STEP 6
415 CALL HCHAR(Z,8,61)
420 CALL HCHAR(Z,9,62)
425 NEXT Z
430 FOR Z=1 TO 1000
435 NEXT Z
440 CALL CLEAR
445 PRINT "VIEL SPASS BEIM SPIELEN!"
450 FOR Z=1 TO 500
455 NEXT Z
460 CALL CLEAR
465 PRINT "NAME>LINKER SPIELER?"
470 PRINT
475 INPUT A$
480 CALL CLEAR
485 PRINT "NAME>RECHTER SPIELER?"
490 PRINT
495 INPUT B$
500 CALL CLEAR
505 A=LEN(A$)
510 B=LEN(B$)
515 FOR Z=1 TO A
520 C$=SEG$(A$,Z,1)
525 C=ASC(C$)
530 CALL HCHAR(2,5+Z,C)
535 NEXT Z
540 FOR Z=1 TO B
545 D$=SEG$(B$,Z,1)
550 D=ASC(D$)
555 CALL HCHAR(2,19+Z,D)
560 NEXT Z
565 CALL HCHAR(4,5,88,28)
570 SC1=0
575 SC2=0
580 RANDOMIZE
585 V=INT(3*RDD)+1
590 IF V=1 THEN 1000
595 IF V=2 THEN 2000
```

```
600 IF V=3 THEN 3000
605 GOTO 580
1000 RANDOMIZE
1005 X=INT((23-5+1)*RND)+5
1010 RANDOMIZE
1015 Y=INT((28-3+1)*RND)+3
1020 CALL HCHAR(X,Y,129)
1025 FOR Z=1 TO 20
1030 CALL KEY(0,K,S)
1035 IF K=ASC("Z") THEN 1060
1040 IF K=ASC("M") THEN 1055
1045 NEXT Z
1047 CALL HCHAR(X,Y,32)
1050 GOTO 580
1055 SC2=SC2-2
1056 CALL HCHAR(X,Y,32)
1057 GOTO 1065
1060 SC1=SC1+1
1061 CALL HCHAR(X,Y,32)
1065 L$=STR$(SC1)
1070 R$=STR$(SC2)
1075 FOR Z=1 TO 2
1076 L$=L$&" "
1077 R$=R$&" "
1080 L1$=SEG$(L$,Z,1)
1085 L1=ASC(L1$)
1090 R1$=SEG$(R$,Z,1)
1095 R1=ASC(R1$)
1100 CALL HCHAR(3,6+Z,L1)
1105 CALL HCHAR(3,22+Z,R1)
1110 NEXT Z
1115 IF SC1>=15 THEN 4000
1120 IF SC2>=15 THEN 4500
1125 GOTO 580
2000 RANDOMIZE
2005 X=INT((23-5+1)*RND)+5
2010 RANDOMIZE
2015 Y=INT((28-3+1)*RND)+3
2020 CALL HCHAR(X,Y,128)
2025 FOR Z=1 TO 20
2030 CALL KEY(0,K,S)
2035 IF K=ASC("M") THEN 2070
2040 IF K=ASC("Z") THEN 2060
2045 NEXT Z
2050 CALL HCHAR(X,Y,32)
2055 GOTO 580
2060 SC1=SC1-2
2061 CALL HCHAR(X,Y,32)
2065 GOTO 2075
2070 SC2=SC2+1
2071 CALL HCHAR(X,Y,32)
2075 GOTO 1065
3000 RANDOMIZE
3005 X=INT((23-5+1)*RND)+5
3010 RANDOMIZE
3015 Y=INT((28-3+1)*RND)+3
3020 CALL HCHAR(X,Y,130)
3025 FOR Z=1 TO 20
3030 CALL KEY(0,K,S)
3035 IF K=ASC("M") THEN 3070
```

```
3040 IF K=ASC("Z") THEN 3060
3045 NEXT Z
3050 CALL HCHAR(X,Y,32)
3055 GOTO 580
3060 SC1=SC1+1
3061 CALL HCHAR(X,Y,32)
3065 GOTO 3075
3070 SC2=SC2+1
3071 CALL HCHAR(X,Y,32)
3075 GOTO 1065
4000 CALL CLEAR
4005 PRINT A$;" HAT GEWONNEN!!!!"
4010 PRINT
4015 PRINT
4020 PRINT "NOCHMAL (J/N)?"
4025 CALL KEY(0,K,S)
4030 IF S=0 THEN 4025
4035 IF K=ASC("J") THEN 10
4040 END
4500 CALL CLEAR
4505 PRINT B$;" HAT GEWONNEN!!!!"
4510 PRINT
4515 PRINT
4520 GOTO 4020
```

3.6 KFZ-Kostenrechnung

Mit Hilfe dieses Programmes ist es Ihnen Möglich, die anfallenden Kosten Ihres KFZ zu ermitteln.
Die Kostenrechnung kann jeweils monatlich durchgeführt werden. Nach den Berechnungen können die Daten auf Kassettenrekorder überspielt werden. Bei diesem Vorgsng werden stets alle bisher errechneten Monatsdaten abgespeichert, so daß Sie alle Daten eines Jahres in einem einzigen Datensatz vereinigt haben.

Wenn Sie das Programm eingegeben und mit RUN gestartet haben, erscheint auf dem Bildschirm das Auswahlmenu:

1-> DATEN EINLESEN
2-> DATEN SPEICHERN
3-> DATENEINGABE
4-> REPARATUREINGABE
5-> KOSTENAUSGABE
6-> REPARATURLISTE
7-> PROGRAMMENDE

Auf die eigentliche Bedienung des Programmes braucht nicht weiter eingegangen zu werden, da das Programm mit Ihnen im Dialog arbeitet. Bitte beachten Sie, daß Kosten- und Reparatureingaben zwar getrennt erfolgen, die Reparaturkosten aber zusätzlich bei der Kosteneingabe unter 'REPARATURKOSTEN' mit eingegeben werden müssen.

Programmbeschreibung:

Zeilennummer	Erläuterungen
10 - 20	Dimensionierung der benötigten Datenfelder.
25 - 35	Definition der benötigten Graphikzeichen.
40 - 145	Erzeugen des Auswahlmenu's mit Abfrageroutine.
150 - 155	Programmende.
200 - 250	Daten einlesen.
300 - 360	Daten abspeichern.
400 - 480	Dateneingabe.
600 - 655	Reparatureingabe.
800 - 970	Ausgabe der Monatskosten.
1000-1085	Ausgabe der Reparaturkosten.
2000-2018	Abfrageroutine für den Monat.
2019-2035	Unterprogramm für die Dateneinleseroutine.

Zeilennummer	Erläuterungen
3000-3030	In diesen Zeilen werden die einzelnen Werte ausgerechnet
4000-4010	Hier wird gefragt, von welchem Monat die Daten ausgegeben werden sollen.

```
10 CALL CLEAR
15 DIM A(4),A$(4),GKM(12),BK(12),BL(12),SK(12)
20 DIM B1$(12,5),C(12,5),KMS(12)
25 CALL CHAR(128,"000000FFFF")
30 CALL CHAR(129,"FFFFFFFFFFFFFFFF")
35 CALL CHAR(130,"1010101010101010")
40 PRINT "BITTE WAEHLEN SIE!"
45 PRINT
50 PRINT
55 PRINT "1-> DATEN EINLESEN"
60 PRINT
65 PRINT "2-> DATEN SPEICHERN"
70 PRINT
75 PRINT "3-> DATENEINGABE"
80 PRINT
85 PRINT "4-> REPARATUREINGABE"
90 PRINT
95 PRINT "5-> KOSTENAUSGABE"
100 PRINT
105 PRINT "6-> REPARATURLISTE"
110 PRINT
115 PRINT "7-> PROGRAMMENDE"
120 PRINT
121 PRINT "****************************"
122 PRINT "JEWEILS MIT 'SPACE' ZURUECK"
125 CALL KEY(0,K,S)
130 IF S=0 THEN 125
135 IF K>=ASC("7") THEN 150
140 G=K-48
145 ON G GOTO 200,300,400,600,800,1000
150 CALL CLEAR
155 END
200 CALL CLEAR
205 REM
210 OPEN #1:"CS1",SEQUENTIAL,INTERNAL,INPUT,FIXED 192
215 CALL CLEAR
220 PRINT "DATEN WERDEN EINGELESEN!"
225 FOR Z=1 TO 12
230 INPUT #1:GKM(Z),BK(Z),BL(Z),SK(Z),KMS(Z)
235 NEXT Z
240 GOSUB 2019
245 CLOSE #1
250 GOTO 2000
300 CALL CLEAR
305 OPEN #1:"CS1",SEQUENTIAL,INTERNAL,OUTPUT,FIXED 192
310 CALL CLEAR
315 PRINT "DATEN WERDEN GESPEICHERT!"
320 FOR Z=1 TO 12
325 PRINT #1:GKM(Z),BK(Z),BL(Z),SK(Z),KMS(Z)
330 NEXT Z
332 FOR X=1 TO 12
335 FOR Z=1 TO 5
340 PRINT #1:B1$(X,Z),C(X,Z)
345 NEXT Z
347 NEXT X
350 CLOSE #1
355 CALL CLEAR
360 GOTO 40
```

```
400 CALL CLEAR
405 RESTORE
410 DATA "GEFAHRENE KM:","BENZINKOSTEN:","BENZIN (LTR):","SONSTIGE
KOSTEN:"
415 FOR Z=1 TO 4
420 READ A$(Z)
425 NEXT Z
430 INPUT "MONAT,JAHR?":M,J
431 PRINT
432 INPUT "KM-STAND BEI MONATSANFANG  :   ":KMS(M)
435 FOR Z=1 TO 4
440 PRINT
445 INPUT A$(Z):A(Z)
450 NEXT Z
452 KMS(M)=KMS(M)+A(1)
455 CALL CLEAR
460 GKM(M)=GKM(M)+A(1)
465 BK(M)=BK(M)+A(2)
470 BL(M)=BL(M)+A(3)
475 SK(M)=SK(M)+A(4)
480 GOTO 40
600 CALL CLEAR
601 GOSUB 4000
605 B$="ARTIKEL :"
610 C$="PREIS   :"
615 FOR Z=1 TO 5
620 PRINT
625 INPUT B$:B1$(M,Z)
630 INPUT C$:C(M,Z)
635 IF B1$(M,Z)="0" THEN 645
640 NEXT Z
645 REM
650 CALL CLEAR
655 GOTO 40
800 CALL CLEAR
801 GOSUB 4000
802 GOSUB 3000
805 PRINT "      KFZ-KOSTEN ";M;"/";J
815 PRINT
816 PRINT "KM-STAND :";KMS(M)
817 PRINT "-----------------------------"
820 PRINT "GEFAHRENE KM";TAB(13);":";GKM(M)
825 PRINT "-----------------------------"
830 PRINT "BENZINKOSTEN";TAB(13);":";BK(M);" DM"
835 PRINT "-----------------------------"
840 PRINT "BENZIN (LTR)";TAB(13);":";BL(M)
845 PRINT "-----------------------------"
850 PRINT "KOSTEN/LTR";TAB(13);":";KL;" DM"
855 PRINT "-----------------------------"
860 PRINT "VERBR./100KM";TAB(13);":";KHK
865 PRINT "-----------------------------"
870 PRINT "REPARATURKOSTEN UND WARTUNG"
875 PRINT
880 PRINT "INCL.OELW.ZUENDK.ETC:";SK(M);" DM"
885 PRINT "-----------------------------"
890 PRINT "KOSTEN/KM";TAB(13);":";KEK;" DM"
895 PRINT "=============================="
900 PRINT "GESAMTKOSTEN";TAB(13);":";GK;" DM"
960 CALL KEY(0,K,S)
965 IF S=0 THEN 960
```

```
967 CALL CLEAR
970 GOTO 40
1000 CALL CLEAR
1002 GOSUB 4000
1005 GOSUB 3000
1010 PRINT "  REPARATURLISTE ";M;"/";J
1015 PRINT
1020 PRINT
1025 FOR Z=1 TO 5
1030 PRINT B1$(M,Z);TAB(18);C(M,Z);" DM"
1035 PRINT
1040 NEXT Z
1045 PRINT TAB(18);"========"
1050 PRINT "GESAMTKOSTEN:";TAB(18);GRK;" DM"
1055 PRINT
1060 PRINT "***************************"
1065 PRINT "MIT LEERTASTE WEITER!"
1070 CALL KEY(0,K,S)
1075 IF S=0 THEN 1070
1080 CALL CLEAR
1085 GOTO 40
2000 CALL CLEAR
2005 INPUT "WELCHER MONAT?:":M
2010 CALL CLEAR
2015 GOSUB 3000
2018 GOTO 40
2019 FOR X=1 TO 12
2020 FOR Z=1 TO 5
2025 INPUT #1:B1$(X,Z),C(X,Z)
2030 NEXT Z
2032 NEXT X
2035 RETURN
3000 KL=BK(M)/BL(M)
3005 KHK=(100*BK(M))/GKM(M)
3010 KEK=(SK(M)+BK(M))/GKM(M)
3015 GK=BK(M)+SK(M)
3020 REM
3025 GRK=C(M,1)+C(M,2)+C(M,3)+C(M,4)+C(M,5)
3030 RETURN
4000 INPUT "VON WELCHEM MONAT?:":M
4005 CALL CLEAR
4010 RETURN
```

3.7 Hausnummer

Dieses Spiel kommt ursprünglich vom Kegeln. Man zeichnet für jeden Spieler neun Kästchen an eine Tafel und beginnt reihum mit dem Kegeln. Jeder Spieler muß nun die Anzahl der getroffenen Kegel in eines dieser Kästchen eintragen. Von links nach rechts gelesen ergibt sich nach Spielende eine neunstellige Zahl.

Beispiel: Folgende Werte wurden in die Kästchen (durch rechteckige Klammern dargestellt) eingetragen:

[3]-[8]-[4]-[9]-[6]-[2]-[1]-[5]-[4]

Von links nach rechts gelesen, ergibt sich die Zahl: 384.962.154

Ob diese Zahl (auch als Hausnummer bezeichnet) möglichst groß oder möglichst klein sein soll, bestimmen die Spieler zu Anfang.

Jedes Kästchen darf nur einmal mit einer Zahl belegt werden, so daß die Speiler reiflich überlegen müssen, wo der gerade erreichte Wert einzutragen ist.

In dem Programm wird der Kegelvorgang durch einen Zufallszahlengenerator simuliert, der willkürlich eine Zahl zwischen 1 und 9 ermittelt. Da ja auch beim Kegeln, gerade bei Anfängern vieles Zufall ist, unterscheidet sich der simulierte Wurf von einem echten nur durch zwei Dinge:

1. Es ist keine Kugel zu rollen.
2. Ein Bandenwurf (0 Punkte) ist hier nicht möglich!

Programmbeschreibung:

Zeilennummer	Erläuterungen
5	Dimensionierung der benötigten Felder.
10 - 80	Definition der Graphikzeichen.
90 - 140	Abfrage, ob eine Spielanleitung gewünscht wird.
150 - 190	Eingabe der Spielernamen und Definition der Spielversion.
200 - 490	Der Spielplan wird generiert.
500 - 560	Wurf des Spielers 1.
570 - 590	Eingabefeld Spieler1 löschen.
600 - 670	Wurf des Spielers 2.
680 - 700	Eingabefeld Spieler 2 löschen.
710 - 760	Alle Felder sind belegt.
770 - 990	Welcher Spieler hat gewonnen und Abfrage, ob noch ein Spiel gewünscht wird.

Zeilennummer	Erläuterungen
1000-3040	In diesen Zeilen wird die Zufallszahl erzeugt und die Felder belegt.
3100-3140	Hilfsroutine zur Siegererkennung bei Spielende.
4000-4110	Spielregeln.

```
5 DIM SUM1$(9),SUM2$(9),S1(9),S2(9)
10 CALL CLEAR
20 CALL CHAR(120,"FFFF")
30 CALL CHAR(121,"000000000000FFF")
40 CALL CHAR(122,"1818181818181818")
50 CALL CHAR(123,"FFFF1818181818181818")
60 CALL CHAR(124,"181818181818FFFF")
70 CALL CHAR(125,"F8F8181818181818")
80 CALL CHAR(126,"181818181818F8F8")
90 PRINT "*********HAUSNUMMER*********"
91 PRINT
92 PRINT
93 PRINT
94 PRINT
95 PRINT
100 PRINT "BRAUCHEN SIE ANWEISUNGEN?"
101 PRINT
110 PRINT "('J' ODER 'N'):"
120 CALL KEY(0,K,S)
122 IF S=0 THEN 120
125 IF CHR$(K)<>"J" THEN 126 ELSE 130
126 IF CHR$(K)<>"N" THEN 120
130 IF CHR$(K)="N" THEN 150
140 GOSUB 4000
150 CALL CLEAR
151 PRINT "WELCHE HAUSNUMMER?"
152 PRINT ::"(1=GROSS,2=KLEIN)"
153 CALL KEY(0,K,S)
154 IF S=0 THEN 153
155 IF K-48=1 THEN 158
156 IF K-48=2 THEN 158
157 GOTO 153
158 FLAG3=K-48
159 CALL CLEAR
160 INPUT "NAME SPIELER 1:":NA1$
161 PRINT
162 PRINT
170 INPUT "NAME SPIELER 2:":NA2$
200 CALL CLEAR
210 REM FELD AUSGEBEN
220 CALL SCREEN(9)
230 FOR I=7 TO 20 STEP 13
240 CALL HCHAR(I,3,120,26)
250 CALL HCHAR(I+2,3,121,26)
260 NEXT I
270 FOR I=11 TO 29 STEP 2
280 CALL VCHAR(6,I,122,4)
290 CALL VCHAR(19,I,122,4)
300 NEXT I
310 FOR Z=11 TO 27 STEP 2
320 CALL HCHAR(7,Z,123)
330 CALL HCHAR(9,Z,124)
340 CALL HCHAR(20,Z,123)
350 CALL HCHAR(22,Z,124)
360 NEXT Z
370 FOR I=7 TO 20 STEP 13
380 CALL HCHAR(I,29,125)
390 CALL HCHAR(I+2,29,126)
```

```
400 NEXT I
401 FOR I=1 TO LEN(NA1$)
402 DR=ASC(SEG$(NA1$,I,1))
403 CALL HCHAR(8,2+I,DR)
404 NEXT I
405 FOR I=1 TO LEN(NA2$)
406 DR=ASC(SEG$(NA2$,I,1))
407 CALL HCHAR(21,2+I,DR)
408 NEXT I
409 Z=48
410 FOR I=12 TO 28 STEP 2
420 Z=Z+1
430 CALL HCHAR(6,I,Z)
440 CALL HCHAR(19,I,Z)
450 NEXT I
460 CALL HCHAR(1,3,42,28)
461 S1$="***** KLEINE HAUSNUMMER******"
462 S2$="******HOHE HAUSNUMMER*******"
463 IF FLAG3=2 THEN 464 ELSE 470
464 FOR I=1 TO LEN(S1$)
465 DR=ASC(SEG$(S1$,I,1))
466 CALL HCHAR(2,2+I,DR)
467 NEXT I
468 GOTO 480
470 FOR I=1 TO LEN(S2$)
471 DR=ASC(SEG$(S2$,I,1))
472 CALL HCHAR(2,2+I,DR)
473 NEXT I
480 CALL HCHAR(3,3,42,28)
490 CALL HCHAR(14,3,42,28)
500 DR$=NA1$&"!-ENTER DRUECKEN"
501 FOR I=1 TO LEN(DR$)
502 DR=ASC(SEG$(DR$,I,1))
503 CALL HCHAR(11,2+I,DR)
504 NEXT I
510 CALL KEY(0,K,S)
511 IF S=0 THEN 510
520 FLAG=1
521 GOSUB 1000
530 DR$="ZUFALLSZAHL="&ZAHL$
531 FOR I=1 TO LEN(DR$)
532 DR=ASC(SEG$(DR$,I,1))
533 CALL HCHAR(12,2+I,DR)
534 NEXT I
540 DR$="WELCHES FELD?:"
541 FOR I=1 TO LEN(DR$)
542 DR=ASC(SEG$(DR$,I,1))
543 CALL HCHAR(13,2+I,DR)
544 NEXT I
550 CALL KEY(0,K,S)
551 IF S=0 THEN 550
552 IF K-48<1 THEN 550
553 IF K-48>9 THEN 550
554 F=K-48
560 GOSUB 1500
570 CALL HCHAR(11,3,32,28)
580 CALL HCHAR(12,3,32,28)
590 CALL HCHAR(13,3,32,28)
600 TR=TR+1
610 DR$=NA2$&"!-ENTER DRUECKEN"
```

```
611 FOR I=1 TO LEN(DR$)
612 DR=ASC(SEG$(DR$,I,1))
613 CALL HCHAR(15,2+I,DR)
614 NEXT I
620 CALL KEY(0,K,S)
621 IF S=0 THEN 620
630 FLAG=2
631 GOSUB 1000
640 DR$="ZUFALLSZAHL="&ZAHL$
641 FOR I=1 TO LEN(DR$)
642 DR=ASC(SEG$(DR$,I,1))
643 CALL HCHAR(16,2+I,DR)
644 NEXT I
650 DR$="WELCHES FELD?:"
651 FOR I=1 TO LEN(DR$)
652 DR=ASC(SEG$(DR$,I,1))
653 CALL HCHAR(17,2+I,DR)
654 NEXT I
660 CALL KEY(0,K,S)
661 IF S=0 THEN 660
662 IF K-48<1 THEN 660
663 IF K-48>9 THEN 660
664 F=K-48
670 GOSUB 1500
680 CALL HCHAR(15,3,32,28)
690 CALL HCHAR(16,3,32,28)
700 CALL HCHAR(17,3,32,28)
710 IF TR<9 THEN 500
720 FOR I=10 TO 18
730 CALL HCHAR(I,3,42,28)
740 NEXT I
750 DR$="E N T E R  D R U E C K E N*"
751 FOR I=1 TO LEN(DR$)
752 DR=ASC(SEG$(DR$,I,1))
753 CALL HCHAR(14,3+I,DR)
754 NEXT I
760 CALL KEY(0,K,S)
761 IF S=0 THEN 760
770 CALL SCREEN(3)
780 CALL CLEAR
790 FOR I=1 TO 9
800 SU1$=SU1$&SUM1$(I)
810 SU2$=SU2$&SUM2$(I)
820 NEXT I
830 PRINT NA1$;" HAT DIE NUMMER:"
831 PRINT
832 PRINT
833 PRINT TAB(8);SU1$
840 PRINT
841 PRINT
842 PRINT "----------------------------"
843 PRINT
850 PRINT NA2$;" HAT DIE NUMMER:"
851 PRINT
852 PRINT
860 PRINT TAB(8);SU2$
870 SU1=VAL(SU1$)
871 SU2=VAL(SU2$)
872 IF FLAG3=2 THEN 873 ELSE 875
873 GOSUB 3100
```

```
874 GOTO 910
875 SU1$=""
876 SU2$=""
880 IF SU1=SU2 THEN 881 ELSE 890
881 PRINT
882 PRINT
883 PRINT "SPIEL IST UNENTSCHIEDEN!"
890 IF SU1<SU2 THEN 891 ELSE 900
891 PRINT
892 PRINT
893 PRINT NA2$;" HAT GEWONNEN!!!"
900 IF SU1>SU2 THEN 901 ELSE 910
901 PRINT
902 PRINT
903 PRINT NA1$;" HAT GEWONNEN!!!"
910 FOR I=1 TO 3000
911 NEXT I
912 CALL CLEAR
913 PRINT "NOCHMAL ('J' ODER 'N')?"
914 PRINT
915 PRINT
916 PRINT
920 CALL KEY(0,K,S)
921 IF S=0 THEN 920
930 IF CHR$(K)="N" THEN END
940 FOR I=1 TO 9
950 S1(I)=0
955 S2(I)=0
970 NEXT I
980 CALL CLEAR
981 TR=0
982 FLAG=0
983 F=0
985 GOTO 210
990 END
1000 RANDOMIZE
1010 ZL=INT(9*RND)+1
1020 ZAHL$=STR$(ZL)
1030 RETURN
1500 IF FLAG=2 THEN 1550
1510 FOR I=1 TO 9
1520 IF F=I THEN 1521 ELSE 1530
1521 IF S1(I)=1 THEN 1522 ELSE 1530
1522 GOSUB 3000
1530 NEXT I
1540 IF TF=1 THEN 1541 ELSE 1600
1541 RETURN
1550 TF=0
1560 FOR I=1 TO 9
1570 IF F=I THEN 1571 ELSE 1580
1571 IF S2(I)=1 THEN 1572 ELSE 1580
1572 GOSUB 3000
1580 NEXT I
1590 IF TF=1 THEN 1591 ELSE 1600
1591 RETURN
1600 IF FLAG=2 THEN 1710
1605 F$=ZAHL$
1610 IF F=1 THEN 1611 ELSE 1620
1611 CALL HCHAR(8,12,ASC(F$))
1612 SUM1$(1)=F$
```

```
1613 S1(1)=1
1614 RETURN
1620 IF F=2 THEN 1621 ELSE 1630
1621 CALL HCHAR(8,14,ASC(F$))
1622 SUM1$(2)=F$
1623 S1(2)=1
1624 RETURN
1630 IF F=3 THEN 1631 ELSE 1640
1631 CALL HCHAR(8,16,ASC(F$))
1632 SUM1$(3)=F$
1633 S1(3)=1
1634 RETURN
1640 IF F=4 THEN 1641 ELSE 1650
1641 CALL HCHAR(8,18,ASC(F$))
1642 SUM1$(4)=F$
1643 S1(4)=1
1644 RETURN
1650 IF F=5 THEN 1651 ELSE 1660
1651 CALL HCHAR(8,20,ASC(F$))
1652 SUM1$(5)=F$
1653 S1(5)=1
1654 RETURN
1660 IF F=6 THEN 1661 ELSE 1670
1661 CALL HCHAR(8,22,ASC(F$))
1662 SUM1$(6)=F$
1663 S1(6)=1
1664 RETURN
1670 IF F=7 THEN 1671 ELSE 1680
1671 CALL HCHAR(8,24,ASC(F$))
1672 SUM1$(7)=F$
1673 S1(7)=1
1674 RETURN
1680 IF F=8 THEN 1681 ELSE 1690
1681 CALL HCHAR(8,26,ASC(F$))
1682 SUM1$(8)=F$
1683 S1(8)=1
1684 RETURN
1690 IF F=9 THEN 1691 ELSE 1700
1691 CALL HCHAR(8,28,ASC(F$))
1692 SUM1$(9)=F$
1693 S1(9)=1
1694 RETURN
1700 RETURN
1710 F$=ZAHL$
1720 IF F=1 THEN 1721 ELSE 1730
1721 CALL HCHAR(21,12,ASC(F$))
1722 SUM2$(1)=F$
1723 S2(1)=1
1724 RETURN
1730 IF F=2 THEN 1731 ELSE 1740
1731 CALL HCHAR(21,14,ASC(F$))
1732 SUM2$(2)=F$
1733 S2(2)=1
1734 RETURN
1740 IF F=3 THEN 1741 ELSE 1750
1741 CALL HCHAR(21,16,ASC(F$))
1742 SUM2$(3)=F$
1743 S2(3)=1
1744 RETURN
1750 IF F=4 THEN 1751 ELSE 1760
```

```
1751 CALL HCHAR(21,18,ASC(F$))
1752 SUM2$(4)=F$
1753 S2(4)=1
1754 RETURN
1760 IF F=5 THEN 1761 ELSE 1770
1761 CALL HCHAR(21,20,ASC(F$))
1762 SUM2$(5)=F$
1763 S2(5)=1
1764 RETURN
1770 IF F=6 THEN 1771 ELSE 1780
1771 CALL HCHAR(21,22,ASC(F$))
1772 SUM2$(6)=F$
1773 S2(6)=1
1774 RETURN
1780 IF F=7 THEN 1781 ELSE 1790
1781 CALL HCHAR(21,24,ASC(F$))
1782 SUM2$(7)=F$
1783 S2(7)=1
1784 RETURN
1790 IF F=8 THEN 1791 ELSE 1800
1791 CALL HCHAR(21,26,ASC(F$))
1792 SUM2$(8)=F$
1793 S2(8)=1
1794 RETURN
1800 IF F=9 THEN 1801 ELSE 1810
1801 CALL HCHAR(21,28,ASC(F$))
1802 SUM2$(9)=F$
1803 S2(9)=1
1804 RETURN
1810 RETURN
3000 IF FLAG=2 THEN 3001 ELSE 3010
3001 SPA=15
3002 GOTO 3020
3010 SPA=11
3020 DR$="FELD SCHON BESETZT!"
3021 FOR I=1 TO LEN(DR$)
3022 DR=ASC(SEG$(DR$,I,1))
3023 CALL HCHAR(SPA,3,DR)
3024 NEXT I
3030 CALL KEY(0,K,S)
3031 IF S=0 THEN 3030
3032 IF K-48<1 THEN 3030
3033 IF K-48>9 THEN 3030
3034 F=K-48
3040 RETURN
3100 IF SU1=SU2 THEN 3101 ELSE 3110
3101 DR$="SPIEL IST UNENTSCHIEDEN!"
3102 FOR I=1 TO LEN(DR$)
3103 DR=ASC(SEG$(DR$,I,1)
3104 CALL HCHAR(20,2+I,DR)
3105 NEXT I
3110 IF SU1<SU2 THEN 3111 ELSE 3120
3111 DR$=NA1$&" HAT GEWONNEN!"
3112 FOR I=1 TO LEN(DR$)
3113 DR=ASC(SEG$(DR$,I,1))
3114 CALL HCHAR(20,2+I,DR)
3115 NEXT I
3120 IF SU1>SU2 THEN 3121 ELSE 3130
3121 DR$=NA2$&" HAT GEWONNEN!"
3122 FOR I=1 TO LEN(DR$)
```

```
3123 DR=ASC(SEG$(DR$,I,1))
3124 CALL HCHAR(20,2+I,DR)
3125 NEXT I
3130 SU1=0
3135 SU2=0
3136 SU1$=""
3137 SU2$=""
3140 RETURN
4000 REM ANWEISUNGEN
4010 CALL CLEAR
4020 PRINT "****************************"
4025 PRINT "*********HAUSNUMMER*********"
4027 PRINT "****************************"
4030 PRINT "DIESES SPIEL KOMMT UR-"
4032 PRINT "SPRUENGLICH  VOM KEGELN.ZIEL"
4034 PRINT "IST ES,AUS NEUN EINZELNEN"
4040 PRINT "WUERFEN EINE MOEGLICHST"
4042 PRINT "GROSSE ANZAHL ZUSAMMEN ZU"
4044 PRINT "'BASTELN'."
4050 PRINT "NACH JEDEM WURF WIRD DIE"
4052 PRINT "ERREICHTE PUNKTEZAHL (1-9)"
4054 PRINT "AUF EINE,DER NEUN STELLEN"
4056 PRINT "GESETZT.SIND ALLE STELLEN"
4060 PRINT "BESETZT,SO HAT DERJENIGE"
4061 PRINT "GEWONNEN,DER VON LINKS NACH"
4064 PRINT "RECHTS GELESEN,DIE GROESSTE"
4066 PRINT "(KLEINSTE) ZAHL ERREICHT"
4070 PRINT "HAT."
4080 PRINT "DIE WUERFE WERDEN IM PRO-"
4082 PRINT "GRAMM DURCH EINEN ZUFALLS-"
4084 PRINT "ZAHLENGENERATOR SIMULIERT!"
4090 PRINT "** VIEL SPASS !!!"
4100 CALL KEY(0,K,S)
4105 IF S=0 THEN 4100
4108 CALL CLEAR
4110 RETURN
```

3.8 Mathe-Trainer

Für die kleinsten unter den Computerfreunden (Grundschulalter) ist das folgende Programm gedacht. Überaus lernfördernd trainiert der Computer die vier Grundrechenarten : Multiplikation, Division, Addition und Subtraktion.

Nach Programmstart wird folgendes Menu auf dem Bildschirm erscheinen:

1-> MULTIPLIKATION
2-> DIVISION
3-> ADDITION
4-> SUBTRAKTION
5-> PROGRAMMENDE

Nun kann die entsprechende Kennziffer eingegeben werden, worauf aus dem gewählten Bereich eime Aufgabe gestellt wird. Zuvor muß noch festgelegt werden, bis zu welcher Obergrenze die Zahlen für die Aufgaben zu erzeugen sind. Danach wird auf dem Bildschirm die Aufgabe etwa so gestellt:

Berechne folgenden Ausdruck:

5 x 4 =

Würde jetzt das richte Ergebnis eingegeben, wird der Computer den Rechenkünstler loben und kehrt dann zum Auswahlmenu zurück. Sollte ein falsches Ergebnis eingetippt werden, bekommt man bis zu drei Chancen, das richtige Resultat zu finden. Sind diese drei Versuche nicht ausreichend, wird das korrekte Ergebnis auf dem Bildschirm ausgegeben.

Anmerkung: Beim Dividieren sind, wenn erforderlich, die Ergebnisse auf zwei Stellen zu Runden.

Beispiel: 4.33333 => 4.33
8.255 => 8.26

Programmbeschreibung:

Zeilennummer	Erläuterungen
5	Rücksetzen des Zählers für die Fehlversuche.
10 - 96	Auswahlmenu.
100 - 150	Festlegen der Zahlenobergrenze
200 - 290	Routine für Multiplikation.
300 - 390	Routine für Division.
400 - 490	Routine für Addition.
500 - 590	Routine für Subtraktion.

```
5 D=1
10 CALL CLEAR
15 PRINT TAB(7);"MATHE-TRAINER"
16 PRINT
17 PRINT
18PRINT
19 PRINT
20 PRINT "BITTE WAEHLEN SIE:"
21 PRINT
22 PRINT
23 PRINT
30 PRINT "1-> MULTIPLIKATION"
31 PRINT
40 PRINT "2-> DIVISION"
45 PRINT
50 PRINT "3-> ADDITION"
55 PRINT
60 PRINT "4-> SUBTRAKTION"
62 PRINT
65 PRINT "5-> PROGRAMMENDE"
66 PRINT
67 PRINT
68 PRINT
70 PRINT "BITTE KENNZIFFER EINGEBEN!"
80 CALL KEY(0,K,S)
85 IF S=0 THEN 80
81 IF K-48<1 THEN 80
82 IF K-48>5 THEN 80
90 Q=K-48
95 IF Q=5 THEN 96 ELSE 100
96 END
100 CALL CLEAR
101 PRINT "BIS ZU WELCHER ◆AHLENOBER-"
102 PRINT
103 PRINT "GRENZE MOECHTEST DU RECHNEN";
110 INPUT A
120 R1=INT(RND*A)+1
130 R2=INT(RND*A)+1
140 CALL CLEAR
150 ON Q GOTO 200,300,400,500
200 PRINT "BERECHNE FOLGENDEN AUSDRUCK:"
201 PRINT
202 PRINT
203 PRINT
210 PRINT R1;"X";R2;"=";
215 INPUT E
220 IF E=R1*R2 THEN 260
230 CALL CLEAR
231 PRINT "DU HAST LEIDER FALSCH"
232 PRINT
233 PRINT "GERECHNET!"
240 D=D+1
241 IF D<=3 THEN 252
242 PRINT ::"DU HAST NACH 3 VER-"
243 PRINT ::"SUCHEN DAS RICHTIGE ERGEB-"
244 PRINT ::"NIS NICHT GEFUNDEN. HIER"
245 PRINT ::"IST DIE RICHTIGE LOESUNG:"
246 PRINT ::R1;"X";R2;"=";R1*R2
```

```
247 PRINT
248 PRINT
249 PRINT "MIT IRGENDEINER TASTE WEITER"
250 CALL KEY(0,K,S)
251 IF S=0 THEN 250
251 GOTO 5
252 PRINT "RECHNE NOCHMAL!"
253 PRINT
254 PRINT
255 PRINT
256 GOTO 200
260 CALL CLEAR
261 PRINT "BRAVO!! DU HAST RICHTIG GE-"
262 PRINT
263 PRINT "RECHNET!"
270 PRINT
271 PRINT
272 PRINT
273 PRINT
274 PRINT "MIT IRGENDEINER TASTE WEITER"
275 CALL KEY(0,K,S)
276 IF S=0 THEN 275
290 GOTO 5
300 PRINT "BERECHNE FOLGENDEN AUSDRUCK:"
301 PRINT
302 PRINT
310 PRINT R1;":";R2;"=";
320 INPUT E
330 IF E=R1/R2 THEN 360
335 D=D+1
340 IF D<=3 THEN 351
341 CALL CLEAR
342 PRINT "DU HAST NACH 3 VERSUCHEN"
343 PRINT :: "DIE LOESUNG NICHT GE-"
344 PRINT ::"FUNDEN. DAS ERGEBNIS"
345 PRINT ::"LAUTET:"
346 PRINT ::::R1;":";R2;"=";R1/R2
347 PRINT ::::"MIT IRGENDEINER TASTE WEITER"
348 CALL KEY(0,K,S)
349 IF S=0 THEN 348
350 GOTO 5
351 CALL CLEAR
352 PRINT "RECHNE NOCHMAL:"
353 PRINT ::::
354 GOTO 300
360 CALL CLEAR
361 PRINT "BRAVO!! DU HAST RICHTIG GE-"
362 PRINT
363 PRINT "RECHNET"
370 PRINT
371 PRINT
372 PRINT
373 PRINT "MIT IRGENDEINER TASTE WEITER"
380 CALL KEY(0,K,S)
381 IF S=0 THEN 380
390 GOTO 5
400 PRINT "BERECHNE FOLGENDEN AUSDRUCK:"
401 PRINT
402 PRINT
410 PRINT R1;"+";R2;"=";
```

```
420 INPUT E
430 IF E=R1+R2 THEN 460
431 D=D+1
432 IF D<=3 THEN 450
433 CALL CLEAR
434 PRINT "DU HAST NACH 3 VER-"
435 PRINT ::"SUCHEN DIE LOESUNG NICHT"
436 PRINT :: "GEFUNDEN. DAS ERGEBNIS"
437 PRINT ::"LAUTET:"
438 PRINT ::::R1;"+";R2;"=";R1+R2
440 PRINT :::: "MIT IRGENDEINER TASTE WEITER"
442 CALL KEY(0,K,S)
444 IF S=0 THEN 442
447 GOTO 5
450 CALL CLEAR
451 PRINT "RECHNE NOCHMAL:"
452 PRINT ::
453 GOTO 400
460 CALL CLEAR
461 PRINT "BRAVO!! DU HAST RICHTIG"
462 PRINT ::"GERECHNET."
470 PRINT ::::"MIT IRGENDEINER TASTE WEITER"
471 CALL KEY(0,K,S)
472 IF S=0 THEN 471
490 GOTO 5
500 PRINT "BERECHNE FOLGENDEN"
501 PRINT :: "AUSDRUCK:"
510 PRINT ::::R1;"-";R2;"=";
520 INPUT E
530 IF E=R1-R2 THEN 560
531 D=D+1
532 IF D<=3 THEN 550
532 CALL CLEAR
533 PRINT "DU HAST NACH 3 VERSUCHEN"
534 PRINT ::"DIE LOESUNG NICHT GE-"
535 PRINT :: "FUNDEN. DAS ERGEBNIS"
536 PRINT ::"LAUTET:"
537 PRINT ::::R1;"-";R2;"=";R1-R1
540 PRINT :::::"MIT IRGENDEINER TASTE WEITER"
541 CALL KEY(0,K,S)
542 IF S=0 THEN 541
543 GOTO 5
550 CALL CLEAR
551 PRINT "RECHNE NOCHMAL:"::
555 GOTO 5
560 CALL CLEAR
561 PRINT "BRAVO!! DU HAST RICHTIG GE-"
562 PRINT ::"RECHNET"
570 PRINT :::: "MIT IRGENDEINER TASTE WEITER"
571 CALL KEY(0,K,S)
572 IF S=0 THEN 571
590 GOTO 5
1000 END
```

3.9 Tabellen- oder Ranglistenerstellung

Wie die Überschrift schon verrät, können Sie mit diesem Programm Tabellen erstellen. Ob nun eine Rangliste für einen Golfclub oder eine Rallye zu erstellen ist, dieses Programm erledigt diese, oft mühevolle Arbeit, sehr zuberlässig für Sie.

Wenn Sie das Programm mit RUN starten, werden Ihnen folgende Eingabedaten abverlangt:

```
WETTBEW.ART:
DATUM      :
ORT        :
TEILN.ZAHL :
```

Durch diese Informationen wird das Programm für ein bestimmtes sportliches Ereignis vorbereitet. Danach wird ein kleines Menu erzeugt, durch das Sie festlegen, ob nach fallenden oder steigenden Werten gerechnet werden soll:

```
1-> NACH FALLENDEN WERTEN
2-> NACH STEIGENDEN WERTEN
```

Beispiel 1: Slalom (nach steigenden Werten)

```
1 STENMARK    SWE  1.4223
2 MUELLER     SUI  1.4354
3 KLAMMER     AUT  1.5218
```

Beispiel 2: Speerwerfen (nach fallenden Werten)

```
1 WOLFERMANN GER 83.75
2 TAFELMEIER GER 83.63
3 PERROT     FRA 79.82
```

Im ersten Beispiel wird der mit der kleinsten Zeit gewinnen. Die Zahlen werden mit steigender Platzziffer immer größer.

Im zweiten Beispiel wird der Gewinner den größten Wert (Weite) erreicht haben. Die Zahlen werden mit steigender Platzziffer immer kleiner.

Die Werte werden zwar immer als Zeit abgefragt, können aber auch Punkte oder Meter sein.

Programmbeschreibung:

Zeilennummer	Erläuterungen
10 - 80	Dimensionierung der Array's und Abfrage der Wettbewerbsdaten.
100 - 168	Wahl der Berechnungskriterien
170 - 234	Dateneingabe der einzelnen Teilnehmer.
235 - 670	Berechnung und Ausgabe der Tabelle.
1000-1090	Berechnung nach steigenden Werten.

Zeilennummer	Erläuterungen
2000-2090	Berechnung nach fallenden Werten.
3000-3090	Abfrage, ob die Tabelle errechnet werden soll und Ausgabe der Wettbewerbsdaten.

```
10 CALL CLEAR
20 DIM NAME$(200),ZEIT(200),LA$(200)
30 PRINT TAB(5);"WETTBEW.ART:";35 INPUT WA$
40 PRINT
45 PRINT TAB(5);"DATUM       :";
50 INPUT DT$
55 PRINT
60 PRINT TAB(5);"ORT         :";
65 INPUT ORT$
70 PRINT
75 PRINT TAB(5);"TEILN.ZAHL :";
80 INPUT TLN
90 CALL CLEAR
100 PRINT "ERSTELLUNG DER TABELLE:"
105 PRINT
106 PRINT
107 PRINT
110 PRINT "1-> NACH FALLENDEN WERTEN"
115 PRINT
120 PRINT "2-> NACH STEIGENDEN WERTEN"
125 PRINT
130 PRINT
135 PRINT
140 PRINT
150 CALL KEY(0.K.S)
155 IF S=0 THEN 150
160 KZ=K-48
164 IF KZ<1 THEN 150
168 IF KZ>2 THEN 150
170 CALL CLEAR
175 INPUT "NAME:":NA$
180 PRINT
185 PRINT
190 INPUT "NATION:":NAT$
195  PRINT
200 INPUT "ZEIT:":ZE
210 ARY=ARY+1
215 CALL CLEAR
220 NAME$(ARY)=NA$
221 ZEIT(ARY)=ZE
222 LA$(ARY)=NAT$
230 IF ARY>TLN THEN 231 ELSE 235
231 PRINT "ZU VIELE EINGABEN!"
232 NAME$(ARY)=""
233 ZEIT(ARY)=0
234 LA$(ARY)=""
235 GOSUB 3000
240 ON KZ GOSUB 1000,2000
250 CALL CHAR(100,"FFFF")
255 PRINT "RANG  NAME     NAT   ZEIT"
256 PRINT
257 PRINT
260 CALL HCHAR(24,3,100,28)
280 A=1
290 GOSUB 590
300 A=21
310 GOSUB 500
320 A=1
```

```
330 GOSUB 500
340 A=61
350 GOSUB 500
360 A=81
380 GOSUB 500
380 A=101
390 GOSUB 500
400 A=121
410 GOSUB 500
420 A=141
430 GOSUB 500
440 A=161
450 GOSUB 500
460 A=181
470 GOSUB 500
480 GOTO 170
490 REM UNTERPROGRAMM
500 FOR Z=A TO ARY
510 PRINT TAB(2);Z;TAB(7);NAME$(Z);TAB(17);LA$(Z);TAB(21);INT(ZEIT(Z)*
10000+.5)/10000
540 IF Z=A+19 THEN 560
550 NEXT Z
560 CALL KEY(0,K,S)
565 IF S=0 THEN 560
570 IF ARY<A+20 THEN 170
580 RETURN
590 FOR Z=A TO ARY
600 PRINT TAB(2);Z;TAB(7);NAME$(Z);TAB(17);LA$(Z);TAB(21);INT(ZEIT(Z)*
10000+.5)/10000
630 IF Z=A+19 THEN 650
640 NEXT Z
650 CALL KEY(0,K,S)
655 IF S=0 THEN 650
660 IF ARY<A+20 THEN 170
670 RETURN
1000 REM STEIGENDE WERTE
1010 FOR S1=1 TO ARY
1020 FOR S2=1 TO ARY
1030 IF ZEIT(S1)>ZEIT(S2) THEN 1040 ELSE 1070
1040 X=ZEIT(S1)
1042 X$=NAME$(S1)
1044 X1$=LA$(S1)
1050 ZEIT(S1)=ZEIT(S2)
1052 NAME$(S1)=NAME$(S2)
1054 LA$(S1)=LA$(S2)
1060 ZEIT(S2)=X
1062 NAME$(S2)=X$
1064 LA$(S2)=X1$
1070 NEXT S2
1080 NEXT S1
1085 CALL CLEAR
1090 RETURN
2000 REM FALLENDE WERTE
2010 FOR S1=1 TO ARY
2020 FOR S2=1 TO ARY
2030 IF ZEIT(S1)<ZEIT(S2) THEN 2040 ELSE 2070
2040 X=ZEIT(S1)
2042 X$=NAME$(S1)
2044 X1$=LA$(S1)
2050 ZEIT(S1)=ZEIT(S2)
```

```
2052 NAME$(S1)=NAME$(S2)
2054 LA$(S1)=LA$(S2)
2060 ZEIT(S2)=X
2062 NAME$(S2)=X$
2064 LA$(S2)=X1$
2070 NEXT S2
2080 NEXT S1
2085 CALL CLEAR
2090 RETURN
3000 CALL CLEAR
3005 PRINT "1-> WETERE EINGABEN"
3010 PRINT
3011 PRINT
3012 PRINT "2-> TABELLE ERRECHNEN"
3013 PRINT
3014 PRINT
3015 PRINT
3020 PRINT "BITTE KENNZIFFER EINGEBEN!!"
3030 CALL KEY(0,K,S)
3035 IF S=0 THEN 3030
3037 IF K-48<1 THEN 3030
3038 IF K-48>2 THEN 3030
3040 IF K-48=1 THEN 170
3050 CALL CLEAR
3055 PRINT TAB(2);"WETTBEWERBSWERTUNG"
3060 PRINT
3061 PRINT
3062 PRINT
3063 PRINT TAB(5);WA$
3070 PRINT
3071 PRINT
3072 PRINT
3073 PRINT TAB(5);"AM";DT$
3080 PRINT
3081 PRINT
3082 PRINT
3083 PRINT TAB(5);"IN";ORT$
3090 RETURN
```

3.10 Schiffe versenken

Wer hat während seiner Schulzeit oder bei anderen Gelegenheiten nicht schon einmal dieses unterhaltsame Spiel gespielt ?

Jeder von Ihnen kennt sicherlich die Spielregeln, so daß diese hier nicht näher erläutert werden müssen.

Das abgedruckte Programm hat einige Besonderheiten. auf die im folgenden näher eingegangen wird:

A) Sie können nicht gegen den Computer spielen. Es muß immer ein Mensch als Spielpartner gewählt werden.

B) Alle Schiffspositionen müssen von den Spielern eingegeben werden. Eine Positionierung durch den Computer ist bei diesem Programm nicht möglich.

C) Die Spieler müssen das Spielende selbst erkennen, da der Computer nur den Spielplan verwaltet. Sicherlich werden einige Leser eine solche Routine einbauen wollen!
Überlegen Sie sich doch einmal eine Möglichkeit.

D) Ein kleiner Pfeil zwischen den Spielfeldern zeigt an, wer gerade am Zug ist.

Programmbeschreibung:

Zeilennummer	Erläuterungen
10 - 245	In diesen Zeilen steht die Spielanleitung.
250 - 300	Auswahl für Schiffseingabe oder Spielbeginn.
305 - 465	Positionierung Spieler 1.
600 - 640	Schuß des Spielers 1.
1000-1120	Erzeugung des Spielplanes.
1200-1285	Hilfsroutine für Graphiken.
2000-2025	Treffer für Spieler 1.
3000-3025	Fehlschuß von Spieler 1.
4000-4020	Namenseingabe von Spieler 1.
5000-5175	Positionierung Spieler 2.
6000-6045	Schuß des Spielers 2.
7000-7020	Treffer von Spieler 2.
8000-8020	Fehlschuß von Spieler 2.
9000-9060	Darstellung der Namen.

Anmerkung: Beachten Sie bitte, daß bei Eingabe der Schußkoordinaten zwischen den Zahlen k e i n Komma eingegeben wird.

```
10 CALL CLEAR
15 PRINT "SCHIFFE VERSENKEN"
20 PRINT
25 PRINT
30 PRINT "DIESES ALTBEKANNTE SPIEL"
35 PRINT
40 PRINT "AUS DER SCHULZEIT WIRD"
45 PRINT
50 PRINT "AUF EINEM 10X10 ZELLEN"
55 PRINT
60 PRINT "GROSSEN FELD AUSGETRAGEN."
65 PRINT
70 PRINT "DIE SPIELREGELN DUERFTEN"
75 PRINT
80 PRINT "ALLGEMEIN BEKANNT SEIN."
85 PRINT
90 PRINT "ES WERDEN FOLGENDE SCHIFFE"
95 PRINT
100 PRINT "VERSTECKT:"
105 PRINT
110 PRINT
115 FOR Z=1 TO 1500
120 NEXT Z
125 CALL CLEAR
130 PRINT "2 ZERSTOERER (JE 3 FELDER)"
135 PRINT
140 PRINT "4 KREUZER (JE 2 FELDER)"
145 PRINT
150 PRINT "5 BOOTE (JE 1 FELD)"
155 PRINT
160 PRINT
165 PRINT "DAS EINGEBEN ERFOLGT NACH"
170 PRINT
175 PRINT "DEM GLEICHEN MUSTER WIE DAS"
180 PRINT
185 PRINT "SUCHEN:"
190 PRINT
195 PRINT "1. HORIZONTALE KOORDINATE"
200 PRINT
205 PRINT "2. KOMMA"
210 PRINT
215 PRINT "3. VERTIKALE KOORDINATE"
220 PRINT
225 PRINT "Z.B. 9,1"
230 PRINT
235 PRINT
240 FOR Z=1 TO 1500
245 NEXT Z
250 CALL CLEAR
255 PRINT "BITTE WAEHLEN SIE:"
260 PRINT
265 PRINT
270 PRINT "1-> SCHIFFE EINGEBEN"
275 PRINT
280 PRINT "2-> SCHIFFE VERSENKEN"
285 PRINT
290 CALL KEY(0,KEY,STATUS)
295 IF STATUS=0 THEN 290
```

```
300 IF KEY=2+48 THEN 600
305 CALL CLEAR
310 DIM POR(9,9),POL(9,9)
315 PRINT "SCHIFFE EINGEBEN-SPIELER 1"
320 PRINT
325 GOSUB 4000
330  FOR ZE=1 TO 3
335 PRINT ZE;".ZERSTOERER"
340 PRINT
345 FOR Z=1 TO 3
350 INPUT H,V
355 POR(H,V)=1
360 NEXT Z
365 NEXT ZE
370 PRINT
375 FOR K=1 TO 4
380 PRINT K;+".KREUZER"
385 PRINT
390 FOR Z=1 TO 2
395 INPUT H,V
400 POR(H,V)=1
405 NEXT Z
410 NEXT K
415 PRINT
420 FOR B=1 TO 5
425 PRINT B;".BOOT"
430 INPUT H,V
435 POR(H,V)=1
440 NEXT B
445 CALL CLEAR
450 PRINT "SCHIFFE SIND POSITIONIERT!"
455 FOR Z=1 TO 1000
460 NEXT Z
465 GOTO 5000
600 GOSUB 1000
605 CALL KEY(0,KEY,STATUS)
606 CALL HCHAR(3,17,62)
610 IF STATUS=0 THEN 605
615 H=KEY-48
620 CALL KEY(0,KEY,STATUS)
625 IF STATUS=0 THEN 620
630 V=KEY-48
635 IF POR(H,V)=1 THEN 2000
640 GOTO 3000
1000 CALL CLEAR
1005 S=4
1010 Z=6
1015 S1=4
1020 Z1=6
1025 S2=5
1030 Z2=6
1035 S3=5
1040 Z3=17
1045 S4=15
1050 Z4=7
1055 GOSUB 1200
1060 S=19
1065 Z=6
1070 S1=19
1075 Z1=6
```

```
1080 S2=20
1085 Z2=6
1090 S3=20
1095 Z3=17
1100 S4=30
1105 Z4=7
1110 GOSUB 1200
1115 CALL VCHAR(5,17,88,15)
1117 GOSUB 9000
1120 RETURN
1200 FOR X=0 TO 9
1205 S=S+1
1210 CALL HCHAR(Z,S,X+48)
1215 NEXT X
1220 FOR X=0 TO 9
1225 Z1=Z1+1
1230 CALL HCHAR(Z1,S1,X+48)
1235 NEXT X
1240 FOR X=1 TO 10
1245 Z2=Z2+1
1250 CALL HCHAR(Z2,S2,126,10)
1255 NEXT X
1260 CALL CHAR(150,"FFFFFFFFFFFFFFFF")
1265 CALL HCHAR(Z3,S3,150,10)
1270 CALL VCHAR(Z4,S4,150,10)
1275 CALL COLOR(12,6,1)
1280 CALL COLOR(15,9,9)
1285 RETURN
2000 REM TREFFER ANZEIGEN
2005 Q=H+7
2010 Y=V+5
2015 CALL HCHAR(Q,Y,88)
2020 CALL SOUND(+500,131,1)
2025 GOTO 605
3000 REM FEHLER ANZEIGEN
3005 Q=H+7
3010 Y=V+5
3015 CALL HCHAR(Q,Y,46)
3020 CALL SOUND(+200,523,1)
3025 GOTO 6000
4000 PRINT
4005 PRINT "IHREN NAMEN BITTE"
4010 PRINT
4015 INPUT A$
4020 RETURN
5000 CALL CLEAR
5005 PRINT "SCHIFFE EINGEBEN-SPIELER 2"
5010 PRINT
5015 PRINT
5020 REM
5025 PRINT "IHREN NAMEN BITTE"
5030 PRINT
5035 INPUT B$
5040 FOR ZE=1 TO 3
5045 PRINT ZE;".ZERSTOERER"
5050 PRINT
5055 FOR Z=1 TO 3
5060 INPUT H,V
5065 POL(H,V)=1
5070 NEXT Z
```

```
5075 NEXT ZE
5080 PRINT
5085 FOR K=1 TO 4
5090 PRINT K;".KREUZER"
5095 PRINT
5100 FOR Z=1 TO 2
5105 INPUT H,V
5110 POL(H,V)=1
5115 NEXT Z
5120 NEXT K
5125 PRINT
5130 FOR B=1 TO 5
5135 PRINT B;".BOOT"
5140 INPUT H,V
5145 POL(H,V)=1
5150 NEXT B
5155 CALL CLEAR
5160 PRINT "SCHIFFE SIND POSITIONIERT!"
5165 FOR Z=1 TO 1000
5170 NEXT Z
5175 GOTO 255
6000 CALL KEY(0,KEY,STATUS)
6001 CALL HCHAR(3,17,60)
6005 IF STATUS=0 THEN 6000
6010 H=KEY-48
6015 FOR Z=1 TO 100
6020 NEXT Z
6025 CALL KEY(0,KEY,STATUS)
6030 IF STATUS=0 THEN 6025
6035 V=KEY-48
6040 IF POL(H,V)=1 THEN 7000
6045 GOTO 8000
7000 Q=H+7
7005 Y=V+20
7010 CALL HCHAR(Q,Y,88)
7015 CALL SOUND(+500,131,1)
7020 GOTO 6000
8000 Q=H+7
8005 Y=V+20
8010 CALL HCHAR(Q,Y,46)
8015 CALL SOUND(+200,523,1)
8020 GOTO 605
9000 A1=LEN(A$)
9005 B1=LEN(B$)
9010 FOR Z=1 TO A1
9015 C$=SEG$(A$,Z,1)
9020 C=ASC(C$)
9025 CALL HCHAR(3,22+Z,C)
9030 NEXT Z
9035 FOR Z=1 TO B1
9040 D$=SEG$(B$,Z,1)
9045 D=ASC(D$)
9050 CALL HCHAR(3,6+Z,D)
9055 NEXT Z
9060 RETURN
```

3.11 Hauptstädte

Das folgende kleine Programm ist als ein Hinweis zu verstehen, wie man die kleinsten unter den Computerfreunden zu Lernerfolgen bringen kann. Das Programm ist in seiner Art nicht nur in der vorgestellten Version anwendbar, man kann es in allen denkbaren Lerngebieten einsetzen.

Wenn sich unsere Kleinen nur kurze Zeit mit dem Programm beschäftigt haben, ist schon ein beachtlicher Zuwachs an Wissen erkennbar.

Nur wem das Lernen Spaß macht, lernt wirklich etwas.

Beweisen Sie mit dem Programm ruhig einmal den Skeptikern, daß der Computer auch nützlich sein kann.

Programmbeschreibung:

Zeilennummer	Erläuterungen
10 - 130	Programmeinleitung mit einem kleinen Text.
150 - 170	Rücksetzen der Zählvariablen.
180 - 190	Einlesen des Landes und der Hauptstadt.
195	Sprung zum Programmende, wenn die Variable L$='XXX' ist.

Zeilennummer	Erläuterungen
200 - 240	Fragestellung mit Eingabe der Antwort.
250	Hier wird überprüft, ob die eingegebene Antwort richtig ist.
260	Wenn dreimal eine falsche Antwort eingegeben wurde, wird die Lösung ausgegeben.
270 - 277	Es wurde eine falsche Stadt geraten. Ein neuer Versuch wird eingeleitet.
280 - 295	Nach drei Versuchen wurde die richtige Antwort nicht gefunden.
300 - 370	Es wurde die richtige Stadt eingegeben.
400 - 460	Programmende mit Ausgabe der Anzahl, der gelösten Aufgaben
470 - 720	Hier stehen die Daten.
1000-1030	Unterprogramm zur Bildschirmausgabe.

```
10 CALL CLEAR
15 PRINT TAB(8);"HAUPSTSTAEDTE"
20 PRINT
22 PRINT
24 PRINT
26 PRINT "ICH FRAGE DICH NACH EINIGEN"
28 PRINT
30 PRINT "HAUPTSTAEDTEN VON LAENDERN."
32 PRINT
34 PRINT "DU HAST JEDESMAL 3 VER-"
36 PRINT
38 PRINT "SUCHE, UM DIE RICHTIGE"
40 PRINT
42 PRINT "STADT ZU FINDEN. SOLLTEST"
44 PRINT
46 PRINT "DU AUSNAHMSWEISE MAL"
48 PRINT
50 PRINT "NICHT WISSEN, UM WELCHE"
52 PRINT
54 PRINT "STADT ES SICH HANDELT,"
56 PRINT
58 PRINT "WERDE ICH DIR DIE LOESUNG"
60 PRINT
62 PRINT "VERRATEN."
64 PRINT
66 PRINT
68 PRINT "DRUECKE IRGENDEINE TASTE!"
70 CALL KEY(0,K,S)
72 IF S=0 THEN 70
74 CALL CLEAR
76 PRINT "DU KANNST AUCH MIT EINEM"
78 PRINT
80 PRINT "FREUND UM DIE WETTE RATEN"
82 PRINT
84 PRINT "DENN ICH WERDE DIR DIE"
86 PRINT
88 PRINT "ANZAHL DER STAEDTE, DIE"
90 PRINT
92 PRINT "DU ERRATEN KONNTEST AM"
94 PRINT
96 PRINT "SCHLUSS MITTEILEN."
98 PRINT
100 PRINT
102 PRINT "DRUECKE IRGENDEINE TASTE!"
110 CALL KEY(0,K,S)
120 IF S=0 THEN 110
130 CALL CLEAR
150 SC=0
160 DL=0
170 Z=0
180 RESTORE
190 READ L$,S$
195 IF L$="XXX" THEN 400
200 Z=Z+1
205 CALL CLEAR
208 PRINT "FRAGE";Z
209 PRINT
210 PRINT
```

```
211 PRINT
212 PRINT "WIE HEISST DIE HAUPTSTADT"
213 PRINT
215 PRINT "VON ";L$;
230 INPUT F$
240 DL=DL+1
250 IF F$=S$ THEN 300
260 IF DL>=3 THEN 280
270 PRINT
271 PRINT
272 PRINT "DU HAST LEIDER FALSCH"
273 PRINT
274 PRINT "GERATEN. VERSUCHE NOCHMAL!"
275 FOR I=1 TO 1000
276 NEXT I
277 GOTO 205
280 CALL CLEAR
281 PRINT "DU HAST NACH 3 VERSUCHEN"
282 PRINT
283 PRINT "DIE RICHTIGE STADT NICHT"
284 PRINT
285 PRINT "GEFUNDEN. DIE HAUPTSTADT"
286 PRINT
287 PRINT "VON ";L$
288 PRINT
290 PRINT "HEISST ";S$
291 FOR I=1 TO 1000
292 NEXT I
293 DL=0
295 GOTO 190
300 CALL CLEAR
301 DL=0
302 PRINT "BRAVO!! DU HAST DIE"
303 PRINT
304 PRINT "RICHTIGE STADT GEFUNDEN!"
305 SC=SC+1
310 IF SC=1 THEN 311 ELSE 320
311 GOSUB 1000
312 GOTO 340
320 REM
330 PRINT
331 PRINT
332 PRINT "DU HAST JETZT";SC;"FRAGEN"
333 PRINT
340 PRINT
341 PRINT "RICHTIG BEANTWORTET!"
350 PRINT
351 PRINT
352 PRINT
353 PRINT "DRUECKE IRGENDEINE TASTE!"
360 CALL KEY(0,K,S)
361 IF S=0 THEN 360
370 GOTO 190
400 CALL CLEAR
401 PRINT "ICH HABE KEINE FRAGEN"
402 PRINT
403 PRINT "MEHR, DIE ICH DIR STELLEN"
404 PRINT
405 PRINT "KOENNTE!"
410 PRINT
```

```
415 PRINT
420 PRINT "DU HAST";SC;"FRAGEN RICH-"
425 PRINT
426 PRINT "TIG BEANTWORTET!"
440 FOR I=1 TO 1000
450 NEXT I
460 END
470 DATA GRIECHENLAND,ATHEN,DEUTSCHLAND,BONN
480 DATA ENGLAND,LONDON,USA,WASHINGTON
490 DATA FRANKREICH,PARIS,DAENEMARK,KOPENHAGEN
500 DATA NIEDERLANDE,DEN HAAG,BELGIEN,BRUESSEL
510 DATA SUED-AFRIKA,PRETORIA,KENIA,NAIROBI
520 DATA CHINA,PEKING,JAPAN,TOKIO
530 DATA JUGOSLAWIEN,BELGRAD,OESTERREICH,WIEN
540 DATA ITALIEN,ROM,ISRAEL,JERUSALEM
550 DATA UDSSR,MOSKAU,SPANIEN,MADRID
560 DATA PORTUGAL,LISSABON,BRASILIEN,BRASILIA
570 DATA KANADA,OTTAWA,NORWEGEN,OSLO
580 DATA SCHWEDEN,STOCKHOLM,FINNLAND,HELSINKI
590 DATA ALBANIEN,TIRANA,ARGENTINIEN,BUENOS AIRES
600 DATA AUSTRALIEN,CAMBERRA,BULGARIEN,SOFIA
610 DATA SCHWEIZ,BERN,IRLAND,DUBLIN
620 DATA INDIEN,DELHI,TUERKEI,ANKARA
630 DATA MAROKKO,RABAT,UNGARN,BUDAPEST
640 DATA CSSR,PRAG,POLEN,WARSCHAU
650 DATA RUMAENIEN,BUKAREST,THAILAND,BANGKOK
660 DATA AEGYPTEN,KAIRO,SYRIEN,DAMASKUS
670 DATA LIBYEN,TRIPOLIS,LIBANON,BAYRUT
680 DATA AFGHANISTAN,KABUL,IRAK,BAGDAD
690 DATA IRAN,TEHERAN,PERU,LIMA
700 DATA TUNESIEN,TUNIS,URUGUAY,MOTEVIDEO
710 DATA VENEZUELA,CARACAS,VIETNAM,HANOI
720 DATA XXX,XXX
1000 PRINT
1010 PRINT
1020 PRINT "DU HAST JETZT 1 FRAGE"
1030 RETURN
```

3.12 Telefonregister

Ein elektronisches Telefonbuch auf Cassette können Sie sich mit diesem Programm anlegen. Als Daten werden die Telefonnummern und die Namen gespeichert. Es können bis zu 100 Datensätze eingegeben werden (die genaue Kapazität können Sie nach dem, in Kapitel 2.2 beschriebenen Verfahren ermitteln). Wenn Ihnen die Kapazität in der Grundausstattung nicht ausreicht, bietet die Speichererweiterungskarte 32 K weiteren Speicherplatz. Um die Kapazität des Programmes zu verändern, können Sie die DIM-Anweisung in Zeile 20 variieren.

Nach Eingabe von RUN meldet sich das Programm mit folgendem Menu:

1-> DATEN EINGEBEN
2-> DATEN SUCHEN
3-> DATEN AUFLISTEN
4-> DATEN ABSPEICHERN
5-> DATEN EINLESEN
6-> PROGRAMMENDE

Wahl '1' -- Im Eingabemodus können Sie Ihre Daten eingeben. Nachdem ein Datensatz eingegeben ist, besteht die Möglichkeit, die eingegebenen Daten zu korrigieren.

Beachten Sie bitte, daß bei der Dateneingabe k e i n e Kommas benutzt werden.

Eingabebeispiele:

NAME:? MEIER-JOSEF

TEL.:? 02411/18623

R I C H T I G

NAME:? MEIER,JOSEF

TEL.:? 02411,18623

F A L S C H

Wahl '2' -- Hier genügt es, wenn Sie einfach einen, Ihnen bekannten Teilstring eingeben. Sie müssen lediglich spezifizieren, ob ein Name oder eine Telefonnummer gesucht werden soll. Im Suchmodus werden alle Datensätze aufgelistet, die irgendwo den Suchstring beinhalten.

Beispiel: Sie suchen nach 'KRA', dann kann die Auflistung so aussehen:

KRAFT

MYKRALLOS

DEKRA

Es können natürlich keine Daten aufgelistet werden, die nicht gespeichert sind.

Wahl '3' -- Alle erfassten Datensätze werden Bildschirmseitenweise aufgelistet.

Unter den Wahlmöglichkeiten '4' und '5' können Sie Ihre Daten mit dem Kassettenrekorder verwalten.

Programmbeschreibung:

Zeilennummer	Erläuterungen
10 - 122	Kurze Anleitung zum Programm.
124 - 230	Auswahlmenu.
240 - 250	Programmende.
260 - 410	Dateneingabe.
1000-1690	Daten suchen.
2000-2060	Daten auflisten.
3000-3070	Daten abspeichern.
4000-4070	Daten einlesen.
10000-11040	Hilfsroutinen 'Tastaturabfrage'
20000-20200	Unterprogramm zum Korrigieren der eingegeben Daten.

```
10 CALL CLAER
12 PRINT TAB(6);TELEFONREGISTER"
20 DIM TN$(100),NA$(100)
30 A$="NAME:"
40 B$="TEL.:"
50 PRINT ::::"DAS FOLGENDE PROGRAMM"
52 PRINT
54 PRINT "SUCHT IHNEN ZUVERLAESSIG"
56 PRINT
58 PRINT "UND SCHNELL GESPEICHERTE"
60 PRINT
62 PRINT "TELEFONNUMMERN HERAUS. SIE"
64 PRINT
66 PRINT "KOENNEN BIS ZU 100 DATEN-"
68 PRINT
70 PRINT "SAETZE ERFASSEN UND DIE"
72 PRINT
74 PRINT "GESPEICHERTEN DATEN MIT DEM"
76 PRINT
78 PRINT "KASSETTENREKORDER FEST-"
80 PRINT
82 PRINT "HALTEN."
84 PRINT
86 PRINT
88 PRINT
90 PRINT "DRUECKE EINE TASTE!"
120 CALL KEY(0,K,S)
122 IF S=0 THEN 120
130 CALL CLEAR
140 PRINT "BITTE WAEHLEN SIE:"
142 PRINT
144 PRINT
146 PRINT
148 PRINT "1-> DATEN EINGEBEN"
150 PRINT
152 PRINT "2-> DATEN SUCHEN"
154 PRINT
156 PRINT "3-> DATEN AUFLISTEN"
158 PRINT
160 PRINT "4-> DATEN ABSPEICHERN"
162 PRINT
164 PRINT "5-> DATEN EINLESEN"
166 PRINT
168 PRINT "6-> PROGRAMMENDE"
210 PRINT
212 PRINT
214 PRINT "BITTE KENNZIFFER EINGEBEN!"
220 CALL KEY(0,K,S)
222 IF S=0 THEN 220
224 IF K<49 THEN 220
226 IF K>54 THEN 220
230 ON K-48 GOTO 260,1000,2000,3000,4000,240
240 CALL CLEAR
245 PRINT "TSCHUESS!!!"
250 END
260 CALL CLEAR
270 FOR I=1 TO 100
275 PRINT "DATENSATZ:";I
```

```
277 PRINT
278 PRINT
280 PRINT A$;
290 INPUT NA$(I)
295 IF NA$(I)="XXX" THEN 296 ELSE 300
296 NA$(I)=""
297 TN$(I)=""
298 GOTO 360
300 PRINT
301 PRINT
302 PRINT B$;
320 INPUT TN$(I)
325 GOSUB 20000
330 CALL CLEAR
340 NEXT I
350 GOTO 370
360 DS=I-1
365 GOTO 130
370 CALL CLEAR
371 FOR I=1 TO 10
372 PRINT
373 NEXT I
374 PRINT "SIE HABEN BEREITS 100"
375 PRINT
376 PRINT "EINGEGEBEN. FUER NEUE"
377 DS=100
378 PRINT
379 PRINT "EINGABEN BITTE EINE NEUE"
380 PRINT
382 PRINT "DATEI EROEFFNEN!"
400 FOR J=1 TO 1000
405 NEXT J
410 GOTO 130
1000 CALL CLEAR
1001 PRINT "SIE KOENNEN NUN DATEN"
1002 PRINT
1003 PRINT "SUCHEN LASSEN. GEBEN SIE"
1004 PRINT
1005 PRINT "EINFACH EINEN IHNEN BE-"
1006 PRINT
1007 PRINT "KANNTEN TEILSTRING EIN. ES"
1010 PRINT
1011 PRINT "BRAUCHT NICHT DIE KOMPLETTE"
1012 PRINT
1013 PRINT "INFORMATION EINGEGEBEN ZU"
1014 PRINT
1015 PRINT "WERDEN, ES MUSS LEDIGLICH"
1016 PRINT
1017 PRINT "SPEZIFIZIERT WERDEN OB SIE "
1020 PRINT
1021 PRINT "EINEN NAMEN ODER EINE TEL.NR."
1022 PRINT
1023 PRINT "SUCHEN."
1030 PRINT
1032 PRINT
1034 PRINT "BEISPIELE:  TAS FUER TASSE"
1036 PRINT
1038 PRINT "              345 FUER 1345"
1080 PRINT
1081 PRINT
```

```
1082 PRINT "DRUECKE EINE TASTE!"
1090 CALL KEY(0,K,S)
1100 IF S=0 THEN 1090
1101 CALL CLEAR
1110 PRINT "ES WERDEN IMMER ALLE"
1111 PRINT
1112 PRINT "DATENSAETZE AUFGELISTET, DIE"
1113 PRINT
1114 PRINT "DIESE SEQUENZ BEINHALTEN."
1130 PRINT
1140 PRINT
1160 PRINT
1170 PRINT "DRUECKE EINE TASTE!"
1180 CALL KEY(0,K,S)
1182 IF S=0 THEN 1180
1190 CALL CLEAR
1192 PRINT "SUCHKRITERIUM:"
1120 PRINT
1201 PRINT
1202 PRINT
1203 PRINT "1-> NAME"
1204 PRINT
1205 PRINT "2-> TEL.NR."
1210 PRINT
1220 PRINT
1221 PRINT "BITTE KENNZIFFER EINGEBEN!"
1230 CALL KEY(0,K,S)
1231 IF S=0 THEN 1230
1232 IF K<49 THEN 1230
1233 IF K>50 THEN 1230
1235 IF K=50 THEN 1600
1240 CALL CLEAR
1245 PRINT A$;
1247 ZL=0
1250 INPUT SU$
1255 CALL CLEAR
1260 SU=LEN(SU$)
1270 FOR I=1 TO 100
1280 FOR J=1 TO LEN(NA$(I))
1290 IF SU$=SEG$(NA$(I),J,SU) THEN 1291 ELSE 1300
1291 GOSUB 1500
1292 ZL=ZL+1
1300 NEXT J
1310 NEXT I
1320 IF ZL>0 THEN 1400
1330 CALL CLEAR
1331 PRINT "DIESE ZEICHENFOLGE IST"
1332 PRINT
1333 PRINT "IN DEN GESPEICHERTEN"
1334 PRINT
1335 PRINT "DATENSAETZEN NICHT ENTHALTEN!"
1360 FOR I=1 TO 1000
1365 NEXT I
1370 GOTO 130
1400 CALL CLEAR
1405 PRINT "KEINE WEITEREN DATEN MIT"
1407 PRINT
1410 PRINT "DIESER FOLGE!"
1410 GOTO 10000
1500 PRINT NA$(I);"->";TN$(I)
```

```
1510 RETURN
1600 CALL CLEAR
1605 PRINT B$;
1610 INPUT SU$
1620 CALL CLEAR
1621 ZL=0
1630 SU=LEN(SU$)
1640 FOR I=1 TO 100
1650 FOR J=1 TO LEN(TN$(I))
1660 IF SU$=SEG$(TN$(I),J,SU) THEN 1661 ELSE 1670
1661 GOSUB 1500
1662 ZL=ZL+1
1670 NEXT J
1680 NEXT I
1690 GOTO 1320
2000 CALL CLEAR
2005 ZR=0
2010 FOR I=1 TO 100
2015 ZR=ZR+1
2020 PRINT NA$(I);"->";TN$(I)
2030 IF ZR=20 THEN 2031 ELSE 2040
2031 GOSUB 11000
2032 ZR=0
2040 IF NA$(I)="" THEN 2041 ELSE 2045
2041 IF TN$(I)="" THEN 2045
2045 NEXT I
2050 GOSUB 11000
2060 GOTO 130
3000 OPEN #1:"CS1",SEQUENTIAL,INTERNAL,OUTPUT,FIXED
3010 PRINT #1:DS
3020 FOR I=1 TO DS
3030 PRINT #1:NA$(I),TN$(I)
3040 NEXT I
3050 CLOSE #1
3060 CALL CLEAR
3070 GOTO 130
4000 OPEN #1:"CS1",SEQUENTIAL,INTERNAL,INPUT,FIXED
4010 INPUT #1:DS
4020 FOR I=1 TO DS
4030 INPUT #1:NA$(I),TN$(I)
4040 NEXT I
4050 CLOSE #1
4060 CALL CLEAR
4070 GOTO 130
10000 PRINT "DRUECKE EINE TASTE!"
10010 CALL KEY(0,K,S)
10020 IF S=0 THEN 10010
10030 GOTO 130
11000 PRINT "DRUECKE EINE TASTE!"
11010 CALL KEY(0,K,S)
11020 IF S=0 THEN 11010
11030 CALL CLEAR
11040 RETURN
20000 PRINT
20010 PRINT "'@'-DRUECKEN FUER AENDERN!"
20020 PRINT
20030 PRINT "KEINE AENDERUNG=> EINE TASTE"
20040 CALL KEY(0,K,S)
20050 IF S=0 THEN 20040
20060 IF K=ASC("@") THEN 20080
```

```
20070 RETURN
20080 CALL CLEAR
20090 PRINT NA$(I);
20100 INPUT TA$
20110 IF TA$="" THEN 20140
20120 NA$(I)=TA$
20140 PRINT
20150 PRINT TN$(I);
20160 INPUT TA$
20170 IF TA$="" THEN 20190
20180 TN$(I)=TA$
20190 CALL CLEAR
20200 RETURN
```

3.13 Minenfeld

Zum Spielen dieses Spieles benötigen Sie eine Fernbedienung (Steuerknüppel).

Auf dem Bildschirm wird ein Feld dargestellt, auf dem bis zu 18 Minen versteckt werden können. Die versteckten Minen (die Anzahl legt einer der Spieler vorher fest) werden für wenige Sekunden gezeigt. In dieser, relativ kurzen Zeit, muß sich der jeweilige Spieler die Lage der Minen merken, um dann mit einem Panzer, der mit Hilfe der Steuerknüppel gelenkt wird, unversehrt ans Ziel zu gelangen. Jedesmal, wenn das Ziel erreicht wird, verbucht der Fahrer einen Punkt. Fährt er jedoch vorher über eine Mine, so wird der Panzer zerstört und der andere Spieler ist an der Reihe. Wer zuerst 10 Punkte erreicht hat ist Sieger.

Dieses Spiel ist vor allem für die Schulung des Gedächtnisses sehr gut geeignet. Das Spiel hat einen kleinen Schwachpunkt, durch den der geübte Spieler leicht einen sicheren Weg zum Ziel finden wird, jedoch möchte ich es Ihnen überlassen, die kleine Schwäche zu entdecken und vielleicht auszumerzen.

Programmbeschreibung:

Zeilennummer	Erläuterungen
100 - 150	Definition der benötigten Graphikzeichen.

Zeilennummer	Erläuterungen
160 - 220	Abfrage, ob eine Spielanleitung gewünscht wird.
230 - 340	Eingabe der Spielernamen und der Anzahl der, zu versteckenden Minen.
350 - 520	In diesen Zeilen wird die Bildschirmgraphik erzeugt.
530 - 630	Überwachung des 1. Panzers.
640 - 680	Explosion des 1. Panzers.
690 - 800	Überwachung des 2. Panzers.
810 - 860	Panzer 2 wurde zerstört.
870 - 990	Welcher Spieler hat gewonnen ?
1000-1130	Verstecken der Minen.
1140-1170	Die Geschwindigkeit eines Panzers wird auf Null gestellt.
1180-1280	Anweisungen.

```
10 REM ********************
20 REM ** SVE - SOFTWARE **
30 REM ********************
100 CALL CLEAR
110 CALL CHAR(105,"FFFFFFFFFFFFFFFF")
120 CALL CHAR(115,"003C7EFFFF7E3C")
130 CALL CHAR(125,"070F1EFCFC1F0F07")
140 CALL CHAR(130,"8552641A99245289")
150 CALL CHAR(135,"1818181818181818")
160 DISPLAY AT(1,8):"MINEN-FELD"
170 DISPLAY AT(10,1):"BRAUCHEN SIE ANWEISUNGEN?"
180 DISPLAY AT(12,1):"('J' ODER 'N')"
190 CALL KEY(0,KY,ST):: IF ST=0 THEN 190
200 IF KY<>74 AND KY<>78 AND KY<>106 AND KY<>110 THEN 190
210 IF KY=78 OR KY=110 THEN 230
220 CALL ANWEISUNG
230 CALL CLEAR
240 DISPLAY AT(5,1):"NAME SPIELER 1:"
250 DISPLAY AT(10,1):"NAME SPIELER 2:"
260 DISPLAY AT(15,1):"WIEVIELE MINEN:"
270 DISPLAY AT(16,1):"(1 BIS 18)"
280 DISPLAY AT(22,1):"NACH EINGABE 'ALPHA LOCK'-"
290 DISPLAY AT(24,1):"TASTE LOESEN!!!!"
300 ACCEPT AT(5,16)BEEP:NA1$
310 ACCEPT AT(10,16)BEEP:NA2$
320 ACCEPT AT(15,16)BEEP:WAHL
330 WAHL=INT(WAHL)
340 IF WAHL<1 OR WAHL>18 THEN 320
350 CALL CLEAR
360 CALL VCHAR(5,3,105,20)
370 CALL VCHAR(5,30,105,20)
380 CALL HCHAR(5,3,105,28)
390 CALL HCHAR(24,3,105,28)
400 CALL VCHAR(6,5,135,7)
410 CALL VCHAR(6,25,135,6)
420 CALL VCHAR(17,5,135,7)
430 CALL VCHAR(17,25,135,7)
440 FOR I=1 TO 4 :: READ C
450 CALL HCHAR(12+I,5,C)
460 NEXT I
470 FOR I=1 TO 5 :: READ C
480 CALL HCHAR(11+I,25,C)
490 NEXT I
500 DISPLAY AT(3,1):NA1$;TAB(14);NA2$
510 DISPLAY AT(1,1):NA1$;" MUSS FAHREN!"
520 GOSUB 1000
530 CALL SPRITE(#15,125,9,112,201)
540 CALL JOYST(1,X1,Y1)
550 CALL MOTION(#15,-2*Y1,2*X1)
560 TR=0
570 CALL COINC(ALL,TR)
580 IF TR<>0 THEN 640
590 CALL POSITION(#15,ZEI,SPA)
600 IF ZEI<32 OR ZEI>192 AND SPA<190 THEN 640
610 IF SPA<40 THEN SCORE1=SCORE1+1 :: GOSUB 1150
615 DISPLAY AT(2,2)SIZE(2):SCORE1 :: IF SCORE1=10 THEN 870
618 GOSUB 1000 :: GOTO 530
620 DISPLAY AT(2,2)SIZE(2):SCORE1 :: IF SCORE1=10 THEN 870
```

```
630 GOTO 540
640 CALL SOUND(1000,-5,0)
650 CALL MOTION(#15,0,0)
660 CALL PATTERN(#15,130)
670 FOR I=1 TO 500 :: NEXT I
680 CALL DELSPRITE(ALL)
690 DISPLAY AT(1,1):NA2$;" MUSS FAHREN!"
700 GOSUB 1000
710 CALL SPRITE(#20,125,3,112,201)
720 CALL JOYST(2,X2,Y2)
730 CALL MOTION(#20,-2*Y2,2*X2)
740 TR=0
750 CALL COINC(ALL,TR)
760 IF TR<>0 THEN 810
770 CALL POSITION(#20,ZEI,SPA)
780 IF ZEI<32 OR ZEI>192 AND SPA <190 THEN 810
790 IF SPA<40 THEN SCORE2=SCORE2+1
795 GOSUB 1150 :: DISPLAY AT(2,15)SIZE(2):SCORE2
798 IF SCORE2=10 THEN 870 :: GOSUB 1000 :: GOTO 710
800 GOTO 720
810 CALL SOUND(1000,-5,0)
820 CALL MOTION(#20,0,0)
830 CALL PATTERN(#20,130)
840 FOR I=1 TO 500 :: NEXT I
850 CALL DELSPRITE(ALL)
860 GOTO 510
870 CALL CLEAR :: IF SCORE1=SCORE2 THEN DISPLAY AT(14,1):"SPIEL IST
UNENTSCHIEDEN!"
880 TR=0
890 CALL DELSPRITE(ALL)
900 IF SCORE1>SCORE2 THEN DISPLAY AT(14,1):NA1$;" HAT GEWONNEN!!!"
910 IF SCORE1<SCORE2 THEN DISPLAY AT(14,1):NA2$;" HAT GEWONNEN!!!"
920 DISPLAY AT(24,1):"NOCHMAL ('J' ODER 'N')?"
930 CALL KEY(0,KY,ST):: IF ST=0 THEN 930
940 IF KY<>74 AND KY<>78 AND KY<>106 AND KY<>110 THEN 930
950 IF KY=78 OR KY=110 THEN END
960 SCORE1,SCORE2=0
970 CALL CLEAR
980 RESTORE
990 GOTO 240
1000 CALL DELSPRITE(ALL)
1010 TR=0
1020 Q=41
1030 FOR I=1 TO WAHL
1040 Q=Q+8
1050 RANDOMIZE :: POSH=INT(RND*(185-41+1))+41
1060 RANDOMIZE :: POSV=INT(RND*(177-41+1))+41
1070 CALL SPRITE(#I,115,2,POSV,Q)
1080 NEXT I
1090 FOR I=1 TO 500 :: NEXT I
1100 FOR I=1 TO WAHL
1110 CALL COLOR(#I,8)
1120 NEXT I
1130 RETURN
1140 DATA 90,73,69,76,83,84,65,82,84
1150 CALL MOTION(#15,0,0,#20,0,0)
1160 CALL SOUND(200,130,0,146,0,164,0)
1170 RETURN
1180 SUB ANWEISUNG
1190 CALL CLEAR
```

```
1200 PRINT "******** MINEN-FELD ********"
1210 PRINT :: PRINT
1220 "IN DIESEM SPIEL HAST DU DIE AUFGABE,EIN MINENFELD ZU
DURCHQUEREN."
1230 PRINT "DU KANNST DIESES SPIEL MIT  EINEM FREUND SPIELEN ODER
AUCH ALLEINE."
1240 ZU ANFANG KANNST DU WAEHLEN,WIEVIELE MINEN IN DEM FELD  VERSTECKT
WERDEN SOLLEN.DER COMPUTER WIRD DIR DIE LAGE  "
1250 PRINT "DER MINEN FUER WENIGE SEKUN-DEN ZEIGEN,IN DENEN DU DIR
EINEN WEG AUSSUCHEN KANNST. DU DARFST SOLANGE FAHREN,BIS"
1260 DU UEBER EINE MINE FAEHRST  ODER 10 PUNKTE ERREICHT HASTJEDESMAL
WENN DU ZUM ZIEL GELANGST,BEKOMMST DU EINEN  PUNKT."
1270 CALL KEY(0,K,S):: IF S=0 THEN 1270 :: CALL CLEAR
1280 SUBEND
```

3.14 Textverarbeitung

Will man seinen Computer sinnvoll ausnutzen, kommt man früher oder später an der Anschaffung eines Textverarbeitungsprogrammes nicht vorbei. Das abgedruckte Programm stellt ein solches dar und ist speziell für die Bildschirmausgabe entwickelt worden.

Die Anwendung des Programmes in Verbindung mit einem Thermodrucker ist ohne weiteres möglich. Wenn Sie jedoch mit grösseren Druckern arbeiten, kann dieses Programm nur nach einigen Änderungen im Ausgabeformat (80 Zeichen/Zeile) verwendet werden.
Wegen der begrenzten Speicherkapazität ist das Programm für die Texteingabe von bis zu 280 Bildschirmzeilen konzipiert. Auch hier können Sie die genaue Kapazitßt des Programmes nach dem, unter 2.2 beschriebenen Verfahren berechnen. Durch verändern der Dimensionierung (30) können Sie die Datenmenge dem, zur Verfügung stehenden Speicherplatz anpassen.

Das Programm ist, wie Sie schnell feststellen werden in EXTENDED BASIC geschrieben und kann auf TI-Basic nicht übertragen werden. Eine vernünftige Textverarbeitung in TI-Basic ist ohnehin nicht möglich, da hier der entsprechende Befehlssatz fehlt.

Nach Programmstart werden Ihnen folgende Auswahlmöglichkeiten angeboten:

1-> TEXT EINGEBEN

2-> TEXT AENDERN

3-> TEXT VERARBEITEN

4-> TEXT ABSPEICHERN

5-> TEXT EINLESEN

6-> PROGRAMMENDE

Wahl '1' -- Hier können Sie Text eingeben. Wenn Sie eine Bildschirmzeile vollgetippt haben, macht ein Pipston darauf aufmerksam, daß die Zeile zu Ende ist. Zum weiterschreiben muß dann 'ENTER' gedrückt werden. Sind Sie mit Ihrer Texteingabe fertig, kann der Eingabemodus durch eingeben von 'XXX' verlassen werden.

Wahl '2' -- Dieser Modus erlaubt es, fehlerhafte Textstellen aufzufinden und zu ändern. Geben Sie einfach den falschen Text ein, dann einen Slash ('/') und danach den richtigen Begriff. Das Programm sucht nun den Fehler und tauscht ihn gegen den korrekten Begriff aus.
Über die Kennziffern '1' (vorwärts blättern) und '2' (rückwärts blättern) können Sie den Text korrektur lesen. Mit Kennziffer '0' verlassen Sie diesen Modus und bekommen das Hauptmenu dargestellt.

Wahl '3' -- In diesem Modus können Sie den eingegebenen Text lediglich lesen. Zum Blättern verwenden Sie bitte die gleichen Kennziffern wie unter Wahl '2' beschrieben.

Wahl '4' -- In diesem Modus wird Text auf einen angeschlossenen Kassettenrekorder überspielt.

Wahl '5' -- Dieser Modus liest Text von einem Kassettenrekorder ein.

Sicherlich werden Sie mit dem Programm erst einmal üben müssen, bis es voll beherrscht wird. Danach werden Sie keine Probleme mehr bei der Anwendung haben.

Programmbeschreibung:

Zeilennummer	Erläuterungen
10 - 160	Hauptwahlmenu.
170 - 330	Texteingabemodul.
2000-2550	Änderung des gespeicherten Textes.
3000-3490	Verarbeitung bzw. lesen des Textes.
4000-4070	Text abspeichern.
5000-5070	Text einlesen.

```
10 CALL CLEAR
20 CALL SCREEN(3)
30 DIM A$(280)
40 DISPLAY AT(1,3)ERASE ALL:"** TEXTVERARBEITUNG **"
50 DISPLAY AT(5,1):"1-> TEXT EINGEBEN"
60 DISPLAY AT(7,1):"2-> TEXT AENDERN"
70 DISPLAT AT(9,1):"3-> TEXT VERARBEITEN"
80 DISPLAY AT(11,1):"4-> TEXT ABSPEICHERN"
90 DISPLAY AT(13,1):"5-> TEXT EINLESEN"
100 DISPLAY AT(15,1):"6-> PROGRAMMENDE"
110 DISPLAY AT(24,1):"BITTE KENNZIFFER EINGEBEN!"
120 CALL KEY(0,K,S) :: IF S=0 THEN 120
130 IF K<49 OR K>54 THEN 120
140 CALL CLEAR
150 ON K-48 GOTO 170,2000,3000,4000,5000,160
160 END
170 CALL SCREEN(9)
180 DISPLAY AT(5,1):"ZEILENABSTAND:"
190 DISPLAY AT(10,1):"1-> OHNE ZWISCHENZEILE"
200 DISPLAY AT(12,1):"2-> MIT ZWISCHENZEILE"
210 DISPLAY AT(24,1):"BITTE KENNZIFFER EINGEBEN!"
220 CALL KEY(0,K,S) :: IF S=0 THEN 220
230 IF K<49 OR K>50 THEN 220
240 CALL CLEAR
250 Z=K-48
260 L=1
270 FOR I=1 TO 280
275 DISPLAY AT(24,1):"'XXX' FUER ENDE EINGEBEN!"
278 DISPLAY AT(L,1):A$(I)
280 ACCEPT AT(L,1)SIZE(-28):A$(I)
285 IF A$(I)="XXX" THEN I=300
290 L=L+Z
300 IF L>21 THEN CALL CLEAR :: L=1
310 NEXT I
320 CALL SCREEN(3)
330 GOTO 40
2000 CALL SCREEN(12)
2010 DISPLAY AT(5,1):"DER TEXT WIRD SEITE FUER"
2020 DISPLAY AT(7,1):"SEITE AUF DEM BILDSCHIRM AUS-"
2030 DISPLAY AT(9,1):"GEGEBEN.SIE HABEN FOLGENDE"
2040 DISPLAY AT(11,1):"OPTIONEN:"
2050 DISPLAY AT(13,1):"1= VORWAERTS BLAETTERN"
2060 DISPLAY AT(15,1):"2= ZURUECK BLAETTERN"
2070 DISPLAY AT(17,1):"0= AENDERUNG BEENDET"
2080 DISPLAY AT(20,1):"WENN SIE FEHLER ENT-"
2090 DISPLAY AT(22,1):"DECKT HABEN,VERFAHREN"
2100 DISPLAY AT(24,1):"MIT LEERTASTE WEITER!"
2110 CALL KEY(0,K,S) :: IF S=0 THEN 2110
2120 DISPLAY AT(5,1)ERASE ALL:"SIE WIE FOLGT:"
2130 DISPLAY AT(7,1):"ZUERST FEHLER EINGEBEN,"
2140 DISPLAY AT(9,1):"DANN SCHRAEGSTRICH UND"
2150 DISPLAY AT(11,1):"DANACH DIE KORREKTUR."
2160 DISPLAY AT(13,1):"BEISPIEL:"
2170 DISPLAY AT(18,1):"FELLER/FEHLER"
2180 DISPLAY AT(24,1):"MIT LEERTASTE WEITER!"
2190 CALL KEY(0,K,S) :: IF S=0 THEN 2190 :: CALL CLEAR
2200 L,T=1
2210 FOR I=1 TO 280
```

```
2230 DISPLAY AT(L,1):A$(T)
2235 L=L+Z :: T=T+1
2240 IF L>21 THEN L=1 :: GOSUB 2280
2250 NEXT I
2260 CALL CLEAR
2270 GOTO 40
2280 DISPLAY AT(22,1):RPT$("*",28)
2290 DISPLAY AT(23,1):"KEIN FEHLER = 'XXX'"
2300 DISPLAY AT(24,1):"FEHLER:"
2310 ACCEPT AT(24,8)BEEP:FEHL$ :: IF FEHL$="XXX" THEN 2460
2311 IF POS(FEHL$,"/",1)=0 THEN 2310
2312 IF FEHL$="" THEN 2310
2313 IF SEG$(FEHL$,POS(FEHL$,"/",1)+1,LEN(FEHL$))="" THEN 2310
2314 IF SEG$(FEHL$,1,POS(FEHL$,"/",1))="/" THEN 2310
2320 REM
2330 Q=POS(FEHL$,"/",1) :: Q1=LEN(FEHL$)
2340 ALT$=SEG$(FEHL$,1,Q-1)
2350 NEU$=SEG$(FEHL$,Q+1,Q1)
2360 FOR J=1 TO 280
2370 IFA$(J)="" THEN 2400
2380 V=POS(A$(J),ALT$,1)
2390 IF V<>0 THEN 2410
2400 NEXT J
2405 GOTO 40
2410 REM
2420 X1=LEN(ALT$) :: X2=LEN(A$(J))
2430 A1$=SEG$(A$(J),1,V-1)
2440 A2$=SEG$(A$(J),V+1,X2)
2450 A$(J)=A1$&NEU$&A2$
2460 DISPLAY AT(24,1):"KENNZIFFER EINGEBEN!"
2470 CALL KEY(0,K,S) :: IF S=0 THEN 2470
2480 IF K<48 OR K>50 THEN 2460
2490 IF K<>48 THEN 2520
2500 I=280
2510 RETURN
2520 IF K<>49 THEN 2540
2530 RETURN
2540 IF Z=1 THEN 2545
2541 IF T>21 THEN T=T-22 :: I=1 :: RETURN
2542 T=1 :: I=1 :: RETURN
2545 IF T>22 THEN T=T-42 :: I=1 :: RETURN
2550 I=1 :: T=1 :: RETURN
3000 CALL CLEAR
 3010 DISPLAY AT(1,5):"TEXT VERARBEITEN"
3020 DISPLAY AT(5,1):"SIE HABEN WAEHREND DER"
3030 DISPLAY AT(7,1):"VERARBEITUNG FOLGENDE OPTI-"
3040 DISPLAY AT(9,1):"ONEN:"
3050 DISPLAY AT(11,1):"1=> VORWAERTS BLAETTERN"
3060 DISPLAY AT(13,1):"2=> ZURUECK BLAETTERN"
3080 DISPLAY AT(17,1):"0=> MENUE"
3090 DISPLAY AT(24,1):"MIT LEERTASTE WEITER!"
3100 CALL KEY(0,K,S) :: IF S=0 THEN 3100 :: IF K<>32 THEN 3100
3110 CALL CLEAR
3120 L,T=1
3125 GOSUB 3950
3160 FOR I=1 TO 22
3170 REM
3180 DISPLAY AT(L,1):A$(T)
3190 L=L+Z :: T=T+1
3200 IF L>21 THEN L=1 :: GOSUB 3220
```

```
3210 NEXT I
3215 GOTO 40
3220 CALL KEY(0,K,S) :: IF S=0 THEN 3220
3230 IF K<48 OR K>50 THEN 3220
3240 IF K=48 THEN 40
3250 IF K=49 THEN I=1 :: RETURN
3260 IF K=50 THEN 3440
3440 IF Z=1 THEN 3480
3450 IF T>21 THEN T=T-22 :: I=1 :: RETURN
3470 I,T=1 :: RETURN
3480 IF T>22 THEN T=T-42 :: RETURN
3490 I,T=1 :: RETURN
3950 DISPLAY AT(22,1):RPT$("*",28)
3960 DISPLAY AT(23,1):"1=> VORWAERTS"
3970 DISPLAY AT(24,1):"2=> RUECKWAERTS"
3990 RETURN
4000 OPEN #1:"CS1",INTERNAL,OUTPUT,FIXED 192
4010 PRINT #1:Z
4020 FOR I=0 TO 270 STEP 10
4030 PRINT #1:A$(I+1),A$(I+2),A$(I+3),A$(I+4),A$(I+5),A$(I+6),A$(I+7),
A$(I+8),A$(I+9),A$(I+10)
4040 NEXT I
4050 CLOSE #1
4060 CALL CLEAR
4070 GOTO 40
5000 OPEN #1:"CS1",INTERNAL,INPUT,FIXED 192
5010 INPUT #1:Z
5020 FOR I=0 TO 270 STEP 10
5030 INPUT #1:A$(I+1),A$(I+2),A$(I+3),A$(I+4),A$(I+5),A$(I+6),A$(I+7),
A$(I+8),A$(I+9),A$(I+10)
5040 NEXT I
5050 CLOSE #1
5060 CALL CLEAR
5070 GOTO 40
```

3.15 Basisumwandlung

Dieses Umrechnungsprogramm ist eine kleine Hilfe für Assemblerprogrammierer. In der Programmierung mit Maschinensprache ist man sehr oft gezwungen, mit Zahlen aus dem Hexadezimalen Zahlensystem zu arbeiten.

Mit der folgenden Routine können Sie Zahlen von maximal 7 Stellen umrechnen. Das Programm ermöglicht es, Zahlen aus dem Dezimalsystem ins Hexadezimalsystem umzuwandeln und umgekehrt.

Nach Programmstart mit RUN meldet sich der Computer mit einer kleinen Wahlmöglichkeit:

VON WELCHER BASIS ?

1=> DEZIMAL
2=> HEXADEZIMAL

BITTE KENNZIFFER EINGEBEN!

Nun müssen Sie lediglich die entsprechende Kennziffer eintippen, danach verlangt der der Computer die umzuwandelnde Zahl, um dann ins gewünschte Zahlensystem umzurechnen.

Beispiel: Sie möchten die Dezimalzahl 154 umrechnen. Tippen Sie '1' ein, um die Zahl 145 eingeben zu können. Nach betätigen von 'ENTER' erscheint das Ergebnis in Form der hexadezimalen Zahl '9A'.

Programmbeschreibung:

Zeilennummer	Erläuterungen
5 - 90	Programmeinleitung mit Abfrage nach umzurechnendem Zahlensystem.
100 - 310	Umrechnung Hexadezimal => Dezimal.
320 - 530	Umrechnung Dezimal => Hexadezimal.
1000-1140	Abfrage, ob neue Berechnung gewünscht wird.

```
5 DIM B(20)
10 CALL CLEAR
15 PRINT TAB(6);"BASISUMWANDLUNG"
20 PRINT
22 PRINT
24 PRINT "DIESES PROGRAMM RECHNET"
26 PRINT
28 PRINT "ZAHLEN AUS DEM DEZIMALEN"
30 PRINT
35 PRINT "ZAHLENSYSTEM INS HEXADEZI-"
40 PRINT
45 PRINT "MALE ZAHLENSYSTEM UND"
47 PRINT
50 PRINT "UMGEKEHRT UM."
52 PRINT
54 PRINT
56 PRINT
58 PRINT "VON WELCHER BASIS?"
60 PRINT
62 PRINT
64 PRINT "1=> DEZIMAL"
66 PRINT
70 PRINT "2=> HEXADEZIMAL"
72 PRINT
74 PRINT
76 PRINT "BITTE KENNZIFFER EINGEBEN!"
80 CALL KEY(0,K,S)
82 IF S=0 THEN 80
84 IF K<49 THEN 80
86 IF K>50 THEN 80
90 IF K=49 THEN 320
100 CALL CLEAR
110 INPUT "UMZUWANDELNDE ZAHL:":X$
120 PRINT
125 PRINT
130 HL=0
140 G=0
150 REM
160 FOR I=1 TO LEN(X$)
170 HX$=SEG$(X$,I,1)
180 H=ASC(HX$)
190 HL=LEN(X$)-I
200 IF H<=57 THEN 230
210 B(I)=(H-55)*(16 HL)
220 GOTO 240
230 B(I)=(H-48)*(16 HL)
240 NEXT I
250 FOR I=1 TO LEN(X$)
260 G=G+B(I)
270 NEXT I
280 PRINT
281 PRINT
282 PRINT
290 PRINT
291 PRINT
292 PRINT
293 PRINT "DEZIMAL =";G
300 PRINT
```

```
310 GOTO 1000
320 CALL CLEAR
322 INPUT "UMZUWANDELNDE ZAHL:":A
330 PRINT
335 PRINT
340 Z=16
345 PRINT
346 PRINT
347 PRINT
348 PRINT
349 PRINT "HEXADEZIMAL =";
360 N=0
370 N=N+1
380 B=INT(A/Z)
390 C=A-(B*Z)
400 B(N)=INT(C+.5)
410 A=B
420 IF B>0 THEN 370
430 GOTO 460
440 N=N-1
450 IF N=0 THEN 520
460 IF B(N)>=10 THEN 500
470 A1=B(N)+48
480 PRINT CHR$(A1);
490 GOTO 440
500 A1=B(N)+55
510 GOTO 480
520 PRINT
530 REM
1000 CALL CLEAR
1010 PRINT "EINE ANDERE ZAHL?"
1020 PRINT
1030 PRINT
1040 PRINT TAB(11);"(J/N)"
1050 PRINT
1060 PRINT
1070 PRINT
1080 PRINT
1090 CALL KEY(0,K,S)
1100 IF S=0 THEN 1090
1110 IF K=ASC("J") THEN 1140
1120 CALL CLEAR
1125 PRINT "TSCHUESS!!"
1130 END
1140 GOTO 10
```

KAPITEL 4: DATENVERWALTUNG MIT DEM TI 99/4A

Einführung

Wer einen Computer besitzt wird früher oder später mit dem Problem der Datenverwaltung konfrontiert werden. Eine große Menge an Daten zu speichern und schnell wieder aufzufinden ist die Hauptanforderung an die nicht immer einfache Programmierung solcher Programme. Wenn man die begleitenden Handbücher nach diesem Thema durchforstet, ist meist nur sehr wenig oder garnichts über die Verarbeitung von Dateien beschrieben. Dies ist sehr verwunderlich, denn gerade die Datenverwaltung ist eines der Hauptanwendungsgebiete von Computern.

In diesem Kapitel wollen wir versuchen, Ihnen einen kleinen Einblick in die Datenverwaltung zu vermitteln. Zum besseren Verständnis wollen wir durch einige Beispiele zeigen, wie die Programmierung einer Datenverwaltung angegangen werden kann. Ein komplett aufgelistetes Dateiverwaltungsprogramm soll Ihnen als Anhalt für Ihre weitere Programmierung dienen. Sicherlich wird jeder von Ihnen eigene Vorstellungen für seine Datenverwaltung haben, die er problemlos in das abgedruckte Listing einbinden kann. Bevor wir mit der eigentlichen Programmierung beginnen wollen, ist es erforderlich, einige Grundbegriffe aus dem Bereich der Datenverwaltung zu klären.

DATEI

Immer häufiger höhrt man in Verbindung mit Computern und Datenverarbeitungsprogrammen den Begriff 'DATEI'. Stellt sich die Frage: Was ist eigentlich eine Datei ?

Am schnellsten und einfachsten erklären wir diese Begriff, indem wir ihn einfach durch einen anderen Begriff ersetzen, den jeder von Ihnen kennt: K A R T E I.
Bekanntlich besteht eine Kartei aus einer Ansammlung von Karteikarten, die zumeist in einem Karteikasten geordnet sind. Solche Karteien enthalten gleichartige Daten, wie z.B. die Buchtitel einer Bücherei mit allen erforderlichen Informationen wie Autor, Bestellnummer und was man sonst noch für wissenswert hält. Man wird zweckmäßigerweise für jeden Titel eine eigene Karteikarte anlegen. Alle Karten zusammen bilden nun eine Kartei. Eine solche Kartei wird nach bestimmten Kriterien geordnet (alphabetisch, nach Artikelnummern oder anderen Gesichtspunkten), um möglichst schnell einen Zugriff auf einzelne Informationen zu haben.
In der EDV spricht man nun von Datei. Ob Kartei oder Datei, beide Begriffe meinen das gleiche. Anstatt von Karteikarte, spricht man nun von Datensatz. Als Karteikasten können wir uns den Computer vorstellen.

Eine Datei ist gegenüber einer Kartei wesentlich flexibler. Ein sehr großer Zeitgewinn beim Suchen und Sortieren der einzelnen Datensätze ist oft genug das Argument, von einer Kartei auf eine Datei umzusteigen. Nicht zuletzt die Platzersparnisse bei einer Datei ist Anlaß, die alte Kartei in der Versenkung verschwinden zu lassen. Die verwendeten Speichermedien (Datenträger) von Kleincomputern sind heute in der Lage, teilweise bis zu 1 Million Zeichen zu speichern. Versuchen Sie sich einmal vorzustellen, wieviele Karteikar-

ten man für die gleiche Kapazität anlegen müßte. Durch einfaches und schnelles Austauschen der Datenträger ist der sekundenschnelle Zugriff auf mehrere Millionen Informationen möglich geworden.

Datensatz

Wie oben schon beschrieben, kann man einen Datensatz mit der Karteikarte einer Kartei vergleichen. Im Datensatz sind alle benötigten Informationen über die jeweilige Sache gespeichert. Wenn ein solcher Datensatz mehrere Informationen enthält, werden diese in sogenannten Feldern gespeichert. Ein Datensatz, der aus eine Adressdatei stammt und als Informationen Namen, Straße und Wohnort enthält, besteht aus drei Feldern.

Felder

Auch hier können wir uns an der Karteikarte orientieren, wo die einzelnen Informationen je einem Feld entsprechen. Die drei, bisher beschriebenen Begriffe kann man sich folgendermaßen im Zusammenhang vorstellen:

DATEI => DATENSATZ => FELD

Wollen Sie auf eine bestimmte Information über eine Person oder eine Sache zugreifen, muß zuerst die entsprechende Datei angesprochen werden. Aus dieser Datei muß nun der betreffende Datensatz gefunden werden. Nun kann man aus diesem

Datensatz das gewünschte Feld lesen. Ein kleines graphisches Schema soll dies verdeutlichen:

DATEI: KUNDENDATEI

	FELD	FELD	FELD	FELD
	NAME	STRASSE	WOHNORT	TELEFON
SATZ 1:	KIRCHNER	JAHNSTRASSE	LEISTADT	1234567
SATZ 2:	PFAHLER	HINTERBERG	DUEW	7654321
SATZ 3:	SCHWINN	NEUMAYERSTR	WEISENHEIM	5634226

.
.
.

Hier können Sie sehr deutlich den Zusammenhang zwischen Datei, Datensatz und Feld erkennen. Wenn Sie eine vernünftige Dateiverwaltung betreiben wollen, sollten Sie die eben erklärten Begriffe verstanden haben.

4.1 Cassette oder Diskette ?

Nach der umfangreichen Einführung wollen wir uns nun mit den verfügbaren Speichermedien beschäftigen. Sicherlich wird die Mehrzahl der Leser über einen Kassettenrekorder verfügen, mit dem Programme oder Daten gespeichert werden. Eine weitere Möglichkeit, Daten zu speichern ist die Diskette in Verbindung mit einer Diskettenstation.

Um die unterschiedlichen Eigenschaften dieser Medien herauszustreichen, wollen wir zunächst eine kleine Rückblende in die Kinderstube der EDV vornehmen.

Begriffe wie 'FLOPPY' oder 'DISKETTE' waren noch vor kurzer Zeit gänzlich unbekannt. Die Computeranwender aus dieser Zeit wollten oder mußten natürlich auch ihre Daten und Programme abrufbereit speichern und waren auf andere Datenträger angewiesen. Ganz zu Anfang der Datenspeicherung hat man die sogenannte Lochkarte entwickelt, auf der ein bestimmtes Lochmuster eingestanzt wurde, das die Informationen enthielt. Dieses relativ preiswerte und einfache Verfahren, das auch heute noch vielerorts seine Anwendung findet, birgt zwei erhebliche Nachteile in sich: Die beiden, zur Handhabung notwendigen Geräte - Lochkartenleser und Lochkartenstanzer - waren rein mechanisch aufgebaut und daher viel zu langsam. Um die EDV-Anlagen noch schneller und somit effektiver zu machen, mußte man ein anderes Speichermedium finden. Bei der Entwicklung eines neuen Systems wurde der Wunsch vieler Computeranhänger nach rascher und sicherer Datenspeicherung berücksichtigt. Bald lag das Ergebnis vor: Das Magnetband. Somit sind wir schon fast bei dem gleichen Verfahren, das auch bei unseren Kassettenrekordern angewandt wird. Auch heute noch sind bei vielen großen EDV-Anlagen

Magnetbandspeicher im Einsatz.

Wenn man die Grundprinzipien dieser Methode betrachtet, hat man auf den ersten Blick ein recht einfaches Verfahren vor sich. Beim TI 99/4A kann man bis zu zwei Kassettenrekorder anschließen, die über einen OPEN-Befehl und die Optionen 'CS1' bzw. 'CS2' angesprochen werden können:

Für den Rekorder 1 z.B.:

```
OPEN #1:'CS1',SEQUENTIAL,INTERNAL,INPUT,FIXED
```

Wie Sie sehen, reicht die bloße Spezifikation 'CS1' nicht aus. Sie müssen noch einige zusätzliche Informationen über Dateiart, Datenorganisation und Ein-/Ausgabe geben. Das obige Beispiel wird dann programmiert, wenn Daten von einem Programm aus eigelesen werden. Für die Speicherung von Programmen müssen andere Befehlsfolgen gegeben werden, die Sie bitte Ihrem Handbuch entnehmen möchten.

Bitte beachten Sie, daß der Rekorder 2 nur zum Einlesen von Daten bestimmt ist, während mit dem Rekorder 1 sowohl Abspeichern wie auch Einlesen möglich ist. Leider ist es beim TI 99/4A nicht möglich, einer Datei einen besonderen Namen zu geben, wie es bei den meisten andern Systemen der Fall ist. Dieser Umstand zwingt Sie dazu, entweder eine manuelle Liste zu führen oder ein Programm zu schreiben, das als Verzeichnis dient.
Wenn Dateien - auch Files genannt - auf Band gespeichert werden, erfolgt dies immer BIT für BIT nacheinander. Diese Dateien nennt man auch sequentielle Dateien. Bei einer solchen Datei muß immer ein ganzer Block von Daten eingelesen oder überlesen werden, wenn man auf bestimmte Informationen Zugriff haben will. Dieser Umstand birgt verständlicherweise einen gewichtigen Nachteil: Die sehr langsame Zugriffszeit.

Mit fortschreitender Technik wurden die Anforderungen an die Speichermedien immer anspruchsvoller. Man überlegte sich, wie man die Speicherkapazität erhöhen und das Zeitproblem verringern könnte. Mit Entwicklung der Magnetplatte konnte man die Datenspeicherung wesentlich verbessern.

Diese Technik hat sich eigentlich bis heute dominierebd gehalten und wird am häufigsten eingesetzt. Auch TI 99/4A-Besitzer können, mit Anschaffung einer Diskettenstation, diese Technik ausnutzen. Wenn Sie eine Diskette in Händen halten, 'können Sie diese am ehesten mit eine Schallplatte vergleichen. Auch werden die Daten in Spuren organisiert. Allerdings ist das Material, aus dem die Disketten gefertigt sind, magnetisch. Die Speichkapazität dieser Disketten ist ungleich größer, als die der Cassetten. Um mit Cassetten die gleich Kapazität zu erreichen, braucht man 4 - 5 Stück des Typs C 60. Bei einer Datenspeicherung mit Floppy-Laufwerk können Sie auch bestimmte File - oder Programmnamen vergeben. Durch ein bestimmtes Organisationsprogramm (DOS) werden alle Daten anhand der Bezeichnungen automatisch verwaltet. Sie können an Ihren TI mehrere Laufwerke, die alle gleichwertig benutzt werden können, anschließen. Auch hier müssen Sie mit einer besonderen Befehlsfolge das jeweilige Laufwerk ansprechen:

```
OPEN #1:'DSK 1',SEQUENTIAL,INTERNAL,OUTPUT
```

Auch hier entnehmen Sie bitte die zusätzlichen Optionen Ihrem Bedienungshandbuch.

Mit einer Diskettenstation können Sie nicht nur sequentielle Dateien anlegen, sondern auch Relativ- und Direktzugriffsdateien verarbeiten (was es damit auf sich hat, wird später beschrieben).

Wir wollen uns nun mit der Frage, die wir uns Eingangs stellten - Cassette oder Diskette - beschäftigen. Eine Antwort hierauf kann sich jeder einzelne selbst geben, nachdem er sich die nachfolgend aufgeführten Vor- und Nachteile durchgelesen hat:

1. Kosten:

Der Kassettenrekorder ist vom Anschaffungspreis her gesehen, das eindeutig preisgünstigere Speichermedium. Für unter 100 DM können Sie schon ein geeignetes Gerät erwerben, während die Kosten für eine Floppy-Station zur Zeit um ein vielfaches höher liegen. Sie können allerdings davon ausgehen, daß diese Geräte in nächster Zukunft erheblich billiger werden. Dieser Trend ist bei gleichwertigen Produkten anderer Firmen (auch bei TI) schon lange spürbar. Etwas anders sieht es bei den Kosten der Datenträger aus. Hier ist die Diskette im Vorteil. Man muß bedenken, daß eine Diskette, die 170.000 Zeichen speichern kann nur 5 - 6 DM kostet. Wie oben schon erwähnt, braucht man ja mehrere Cassetten, um die gleiche Speicherkapazität zu erreichen. Sie können sich selbst ausrechnen, um wieviel teurer die Datenspeicherung auf Band ist.

2. Zugriffszeit:

Wer bei der Kostenfrage noch Zweifel hegt, muß spätestens jetzt einsehen, daß die Diskette diesbezüglich im Vorteil ist. Vergleicht man die beiden Systeme anhand sequentieller Dateien, stellt sich sehr schnell heraus, daß die Diskette ca. 10 mal schneller ist.

3. Programmierkomfort:

Bei einer Diskettenstation haben Sie die Möglichkeit, zwi-

schen drei verschiedenen Zugriffsarten zu wählen: Neben den sequentiellen Dateien sind Relativ- oder Direktzugriffsdateien programmierbar, die Ihnen ein wesentlich breiteres Spektrum an Möglichkeiten eröffnen.

Einigen Lesern wird klar geworden sein, daß der Kassettenrekorder allenfalls für den Einsteiger interessant ist. Wer seinen Computer voll ausnutzen oder gar kommerziell einsetzen will, kommt an der Anschaffung eines Floppylaufwerkes nicht vorbei. Da wir davon ausgehen, daß die Mehrzahl der Leser aus Kostengründen nur einen Kassettenrekorder besitzen, werden wir im Folgenden nur auf die Handhabung und Programmierung mit Bandgeräten eingehen. Für die stolzen Besitzer eines Floppylaufwerkes ist es sicherlich nicht schwer, alle vorgestellten Programme oder Routinen für die Datenspeicherung mit Diskette umzuschreiben.

4.2 Cassettenoperationen

Wir wollen Ihnen nun zeigen, wie man den Kassettenrekorder zur Speicherung von Daten und Programmen einsetzt. Wie schon erwähnt, können bei der Datenspeicherung auf Band keine Programmnamen vergeben werden. Deshalb ist man dazu gezwungen, sich die Stellen zu merken, an denen die jeweiligen Daten gespeichert sind. Sehr nützliche Dienste kann hier ein Zählwerk tun, das leider nicht immer voehanden ist. Achten Sie deshalb beim Kauf eines Rekorders auf diese Einrichtung.

4.2.1 Speichern von Programmen

Zum Speichern eines Programmes müssen Sie folgende Befehlszeile eingeben:

```
SAVE CS1
```

Wie an anderer Stelle schon beschrieben, muß beim Speichern von Daten immer der Rekorder 1 angesprochen werden, da mit dem zweiten Gerät nur gelesen werden kann. Es ist empfehlenswert, zwischen den einzelnen Programmen einen kleinen Zwischenraum zu lassen, damit beim Einlesen keine Fehler auftreten können. Üblicherweise ist der Leerraum, den der TI von sich aus läßt, völlig ausreichend, doch kann sich bei einer Änderung oder Verbesserung eines Programmes (Programm wird länger) ein zusätzlicher Freiraum auszahlen, wenn das Programm an der gleichen Bandstelle überschrieben werden soll.

ACHTUNG: Beim Abspeichern überprüft der Computer nicht, ob schon Daten auf dem Band sind, sondern würde evtl. vorhandene Daten oder Programme überschreiben !

4.2.2 Einlesen von Programmen

Wenn Sie Programme einlesen wollen können Sie wählen, ob das Programm von Kassettenrekorder 1 oder 2 eingespielt werden soll:

OLD CS1

oder

OLD CS2

Hier wird Ihnen ein Zählwerk sicherlich gute Dienste leisten. Vorausgesetzt: Sie haben Ihre Programmsammlung schriftlich fixiert. Beachten Sie, daß der Rechner die Programme nur dann einliest, wenn der Header (Programm- oder Datenkopf), der bei jedem Speichervorgang zur Kennzeichnung des Fileanfangs auf Band geschrieben wird, vollständig gelesen wird. Spulen Sie Ihr Band also besoders sorgfältig an den Programmanfang.

4.2.3 Speichern von Dateien

Bevor Daten aus einem Programm abgespeichert werden, muß durch eine bestimmte Befehlsfolge eine Datei eröffnet werden (es können mehrere Dateien gleichzeitig geöffnet sein):

OPEN #DN,'Gerätename',Dateiart,Modus,Verarbeitung,Länge

#DN : Dieser Wert kann eine ganze Zahl zwischen 0 und 255 sein und gibt mit vorangestelltem Nummernzeichen (#) die Dateinummer an. Es ist zu beachten, daß eine Dateinummer nicht mehrmals vergeben wird, da sonst eine Fehlermeldung ausgegeben wird. Die Zahl 0 ist für Bildschirm bzw. Tastatur vorgesehen und darf für die Filebehandlung auf externen Geräten n i c h t vergeben werden.

Gerätename : Beim Speichern von Daten ist dies immer der Kassettenrekorder 1 (CS1). Der Gerätename ist stets in Anführungszeichen zu setzen. Sollten Sie eine Floppystation besitzen, muß als Gerätename das Laufwerk eingegeben werden (z.B. 'DSK1' bei Diskettenstation 1)

Dateiart : Hier kann angegeben werden, ob es sich um eine relative oder um eine sequentielle Datei handelt. Da bei der Datenspeicherung auf Band nur sequentielle Dateien angelegt werden können (nur eine Floppy kann relative Dateien verarbeiten), muß hier immer 'SEQUENTIAL' eingegeben werden.

Modus : Der Modus legt fest, ob Daten abgespeichert ('OUTPUT') oder eingelesen ('INPUT') werden.

Verarbeitung : Hier wird festgelegt, ob eine Datei von dem Computer INTERNAL oder im DISPLAY verarbeitet wird:

INTERNAL - Der Computer wandelt alle Daten in den Binärcode um. Da dieser Code von dem Rechner direkt gelesen werden kann, ist eine Verarbeitung im INTERNAL-Modus wesentlich schneller als im DISPLAY-Modus. Für den Anwender ist es nur sehr schwer zu realisieren, Daten im INTERNAL-Modus zu entziffern. Wenn man den ASCII-Code zugrunde legt, kann man sich die Darstellung des Wortes 'COMPUTER' im INTERNAL-Modus etwa so vorstellen:

C -- 0 1 0 0 0 0 1 1
O -- 0 1 0 0 1 1 1 1
M -- 0 1 0 0 1 0 1 0
P -- 0 1 0 1 0 0 0 0
U -- 0 1 0 1 0 1 0 1
T -- 0 1 0 1 0 1 0 0
E -- 0 1 0 0 0 1 0 1
R -- 0 1 0 1 0 0 1 0

In der Regel wird man alle Daten im INTERNAL-MODUS verarbeiten. Auch ist hier die Platzersparnis erheblich größer als im DISPLAY-Modus.

DISPLAY -- Alle Daten werden im ASCII-Code, also für den Anwender direkt lesbar, verarbeitet. Diesen Modus, der mehr Speicher- bzw. Bandplatz benötigt als der INTERNAL-Modus, wird man dort anwenden, wo alle Daten direkt - während dem Einlesevorgang - auf dem Bildschirm dargestellt werden sollen.

Länge : Wenn Sie hier eine Datensatzlänge definieren (z.B. FIXED 100), ist jeder Datensatz gleich

lang. Das heißt, ein Datensatz wird entweder abgeschnitten oder mit Leerzeichen aufgefüllt. Die FIXED-Option sollte man dort anwenden, wo immer gleichartige und gleichlange Eingaben erfolgen.
Wollen Sie Datensätze mit unterschiedlichen Längen verarbeiten, so geben Sie hier VARIABLE ein. Nun brauchen Sie auf eine bestimmte Eingabelänge nicht mehr zu achten.

Sie brauchen nicht alle Optionen zur Eröffnung einer Datei in einer festgelegten Reihenfolge einzugeben, mit Ausnahme von #DN und Gerätename. Wenn Sie einige Optionen im OPEN-Statement weglassen, nimmt der TI 99/4A folgende Standartwerte an:

Dateiart	=>	SEQUENTIAL
Verarbeitung	=>	DISPLAY
Länge	=>	FIXED 64

Beispiel: Sie finden in einem Programm das folgende OPEN-Statement:

```
OPEN #1:'CS1',OUTPUT
```

Von Ihrem Computer wird diese Zeile so behandelt, als stände dort:

```
OPEN #1:'CS1',SEQUENTIAL,DISPLAY,OUTPUT,FIXED 64
```

4.2.4 Einlesen von Daten

Beim OPEN-Statement, das ein File zum Einlesen von Daten öffnet, müssen alle Optionen genauso programmiert werden wie bei dem OPEN-Statement, das ein File zum Abspeichern öffnet. Wenn Sie dies nicht berücksichtigen, kann es zu Lesefehlern kommen. Hier muß unter Modus 'INPUT' eingegeben werden.

Beispiel:

```
10 OPEN #1:'CS1',SEQUENTIAL,INTERNAL,OUTPUT
 .          Das Band wird mit dem Befehl
 .          PRINT #1:  'beschrieben'.
20 CLOSE #1
 .
 .
30 OPEN #1:'CS1',SEQUENTIAL,INTERNAL,INPUT
 .          Die Daten werden mit dem Befehl
 .          INPUT #1:  eingelesen.
40 CLOSE #1
```

Beachten Sie, daß jeder Speicher- oder Lesevorgang mit CLOSE #DN abgeschlossen wird, da sich ein geöffnetes File kein zweites mal öffnen läßt.

Weitere Anwendungen über das OPEN-Statement werden in dem Kapitel 'Der TI und seine Umwelt' beschrieben.

Wenn Sie nun mit allen Cassettenoperationen vertraut sind, können wir endlich mit der Programmierung unseres Datenverwaltungsprogrammes beginnen.

4.3 Sequentielle Datei: Was ist das ?

Bei der Datenspeicherung mit Cassette können die Daten nur nacheinander auf's Band geschrieben oder von Band gelesen werden. Man kann auch sagen: Die Daten sind sequentiell, also nacheinander abgespeichert.

Als einfaches Beispiel kann man hier eine bespielte Musikcassette anführen, auf der die Melodien ja auch nacheinander gespeichert sind. Wenn Sie hier ein bestimmtes Musikstück höhren wollen, müssen Sie mit Hilfe eines Zählwerkes (sofern vorhanden) durch Vor- oder Zurückspulen die Bandstelle ausfindig machen, wo das Stück beginnt.
Wenn Sie nun die PLAY-Taste betätigen, wird über den Tonkopf des Kassettenrekorders Ton für Ton aufgenommen und abgespielt.

Genauso können Sie sich das Abspeichern und Einlesen von Daten oder Programmen vorstellen. Hier wird über den Tonkopf BIT für BIT eingelesen und dem Arbeitsspeicher zugeführt.

Wenn Sie sich das eben erläuterte einmal vor Augen halten, kommen Sie ganz von selbst zu der Erkenntnis, warum auf einem Band nur sequentiell gearbeitet werden kann. Ein kleines Beispiel soll dies noch einmal veranschaulichen:

Tonband

AUF DIESEM BAND SIND DIE DATEN SEQUENTIELL GESPEICHERT !!!!

Wenn Sie dieses Band laufen lassen können Sie sehen, wie die Buchstaben nacheinander (sequentiell) verschwinden.

4.4 Eine kleine Routine zum Schreiben auf Band

Wir wollen nun eine kleine Routine programmieren, über die wir beliebig viele Namen auf Band abspeichern können. Der Speichervorgang soll erst beendet werden, wenn bei der Abfrage 'NAME:' das Wort 'STOP' eingegeben wird.

```
100 REM ***************************************************
110 REM *** SCHREIBEN VON DATEN AUF BAND                 ***
120 REM ***************************************************
130 REM BILDSCHIRM LOESCHEN
140 CALL CLEAR
150 PRINT 'DATEI WIRD EROEFFNET !'
160 PRINT
170 OPEN #1:'CS1',SEQUENTIAL,INTERNAL,OUTPUT,FIXED
180 CALL CLEAR
190 INPUT 'NAME       :':N$
200 INPUT 'VORNAME    :':VN$
210 CALL CLEAR
220 PRINT 'DATEN WERDEN GESPEICHERT !'
230 PRINT #1:N$
240 IF N$='STOP' THEN 270
250 PRINT#1:VN$
260 GOTO 180
270 CALL CLEAR
280 PRINT 'DATEI WIRD GESCHLOSSEN !'
290 CLOSE #1
300 END
```

Sie können nun solange Namen eingeben, bis in Zeile 190 das Wort 'STOP' eingegeben wird. In Zeile 240 wird, nachdem 'STOP' abgespeichert wurde, zur Zeile 270, wo das Programm beendet wird, gesprungen.

4.5 Eine kleine Leseroutine

Als Gegenstück zu dem unter 4.4 beschriebenen Programm soll nun eine Routine programmiert werden, die es uns ermöglicht, die abgespeicherten Daten wieder einzulesen:

```
310 REM ***************************************************
320 REM *** LESEN VON NAMEN VOM BAND                    ***
330 REM ***************************************************
340 REM BILDSCHIRM LOESCHEN
350 CALL CLEAR
360 PRINT 'DATEI WIRD EROEFFNET !'
370 PRINT
380 OPEN #1:'CS1',SEQUENTIAL,INTERNAL,INPUT,FIXED
390 CALL CLEAR
400 INPUT #1:N$
410 IF N$='STOP' THEN 480
420 INPUT #1:VN$
430 PRINT 'NAME        :';N$
440 PRINT 'VORNAME     :';VN$
450 FOR I=1 TO 500
460 NEXT I
470 GOTO 390
480 CALL CLEAR
490 PRINT 'DATEI WIRD GESCHLOSSEN !'
500 CLOSE #1
510 END
```

Die Leseroutine wird solange Namen einlesen und auf dem Bildschirm ausgeben, bis N$ das Wort 'STOP' enthält (Zeile 410). Wenn N$='STOP' wird das File geschloßen (480-510).

4.6 Erweitern einer Datei

Wenn Sie unsere beiden, unter 4.4 und 4.5 beschrieben Routinen zusammenfügen, haben Sie bereits eine kleine Datenverwaltung programmiert. Mit etwas Übung wird es Ihnen sicherlich leicht fallen, durch einige Änderungen eine Adressenverwaltung aufzubauen, mit deren Hilfe alle möglichen Adressen eingegeben und gespeichert werden können. Irgendwann werden Sie Ihre Adressensammlung erweitern wollen (der Bekanntenkreis wird sicherlich immer größer werden) und suchen nach einer Lösung, wie Daten an die bereits bestehenden Datensätze angehängt werden können. In einem kleinen Beispiel wollen wir Ihnen eine mögliche Lösung zeigen, mit der man Dateien auf Band erweitern kann.

Wie Sie wissen, ist es nicht ohne weiteres möglich, bei Verwendung eines Kassettenrekorders als Datenträger, Daten anzufügen. Wir müssen also die gesamte Datei in den Arbeitsspeicher Ihres TI einlesen, sie durch Anfügen von neuen Daten erweitern und schließlich die neu entstandene Datei über den alten Daten abspeichern. Hierbei müssen Sie darauf achten, daß der 16K-Arbeitsspeicher für die erweiterte Datei ausreicht. Beim Abspeichern der neuen Datei müssen Sie besonders vorsichtig sein, da der benötigte Platz auf dem Band entsprechend größer sein wird und unter Umständen nachfolgende Dateien oder Programme ganz oder teilweise überschrieben werden.

Nach dem gleichen Verfahren lassen sich innerhalb einer Datei natürlich auch einzelne Daten ändern bzw. löschen.

Doch nun zu der Routine selbst, die sich an unsere beiden vorangegangenen Beispiele anlehnt un das Anfügen von Namen

und Vornamen auf Band ermöglicht:

```
100 REM ***************************************************
110 REM *** ROUTINE ZUR ERWEITERUNG VON DATEIEN AUF BAND ***
120 REM ***************************************************
130 REM BILDSCHIRM LOESCHEN
140 CALL CLEAR
150 PRINT 'DATEI WIRD ZUM EINLESEN DER'
160 PRINT
170 PRINT 'ALTEN DATEI GEOEFFNET !'
180 PRINT
190 OPEN #1:'CS1',SEQUENTIAL,INTERNAL,INPUT,FIXED
200 CALL CLEAR
210 REM ES KOENNEN MAX. 50 NAMEN EINGEGEBEN WERDEN
220 DIM N$(50),VN$(50)
230 DS=1
240 INPUT #1:N$(DS)
250 IF N$(DS)='STOP' THEN 300
260 INPUT #1:VN$(DS)
270 IF DS=50 THEN 310
280 DS=DS+1
290 GOTO 240
300 DS=DS-1
310 CLOSE #1
320 PRINT 'BAND ZURUECK SPULEN!'
330 PRINT
340 PRINT 'FERTIG (J/N) ?'
350 CALL KEY (0,K,S)
360 IF S=0 THEN 350
370 IF CHR$(K)='J' THEN 390
380 GOTO 350
390 CALL CLEAR
400 PRINT 'OEFFNEN DER DATEI ZUM'
410 PRINT
```

```
420 PRINT 'ERWEITERN !'
430 PRINT
440 OPEN #1:'CS1',SEQUENTIAL,INTERNAL,OUTPUT,FIXED
450 CALL CLEAR
460 FOR I=1 TO DS
470 PRINT #1:N$(I)
480 PRINT #1:VN$(I)
490 NEXT I
500 PRINT 'DATEN ANFUEGEN:'
510 PRINT ::
520 INPUT 'NAME      :':NA$
530 IF NA$='STOP' THEN 610
540 INPUT 'VORNAME  :':VORN$
550 PRINT ::
560 PRINT 'DATEN SPEICHERN'
570 PRINT #1:NA$
580 PRINT #1:VORN$
590 CALL CLEAR
600 GOTO 500
610 PRINT 'DATEI SCHLIESSEN'
620 PRINT ::
630 CLOSE #1
640 END
```

Aus diesen drei Beispielen (4.4, 4.5 und 4.6) können Sie schon einen kleinen Grundstein für ein leistungsfähiges Adressenverwaltungsprogramm legen, das allerdings noch um die fehlenden Daten ergänzt werden muß (STR.,WOHNORT,TEL. u.s.w.).

4.7 Eine universelle Datenverwaltung

Nachdem wir uns bis jetzt mit den Grundlagen der Datenverwaltung beschäftigt haben, wollen wir nun daran gehen, selbst ein solches Programm zu schreiben. Unser Programm soll nicht speziell für einen bestimmten Anwendungszweck einsetzbar sein, wir möchten vielmehr alle unsere Daten damit verwalten können. Bei einer Adressverwaltung zum Beispiel werden die Eingabedaten immer durch die gleichen Abfragekriterien gefordert. Bei einem universellen Programm müssen diese Kriterien durch den Anwender während des Programmlaufes definiert werden können. Das heißt: Wenn für die eine Datei nach Namen, Vornamen, Strasse und Wohnort abgefragt wird, muß das gleiche Programm ohne großen Aufwand für die nächste Datei nach Artikelnummer, Preis und Lagerort fragen können. Es gibt zwei Möglichkeiten, diese Abfragekriterien festzulegen:

1. Die Abfragekriterien werden in DATA-Zeilen abgelegt und über RESTORE N eingelesen. Natürlich muß ein entsprechendes MENU auf dem Bildschirm generiert werden, das nach Eingabe einer Kennziffer für die gewünschte Datenverwaltung den Programmzeiger auf die DATA-Zeile positioniert, in der die benötigten Daten stehen. Beispiel:

```
10 CALL CLEAR
20 PRINT 'BITTE WAEHLEN SIE:'::::
30 PRINT '1=> ADRESSDATEI'::
40 PRINT '2=> LAGERHALTUNG'::
50 PRINT '3=> BIBLIOTHEK'::::
60 PRINT 'BITTE KENNZIFFER EINGEBN!'
70 CALL KEY(0,K,S)
80 IF S=0 THEN 70
```

```
90 IF K-48<1 THEN 70
100 IF K-48>3 THEN 70
110 CALL CLEAR
120 IF K-48<>1 THEN 150
130 RESTORE 900
140 GOTO 190
150 IF K-48<>2 THEN 180
160 RESTORE 910
170 GOTO 190
180 RESTORE 920
190 DIM B$(10)
200 I=1
210 READ B$(I)
220 IF B$(I)='XXX' THEN 250
230 I=I+1
240 GOTO 210
250 BEZ=I-1
260 FOR I=1 TO BEZ
270 PRINT B$(I);
280 INPUT A$
290 PRINT
300 NEXT I
  .
  .      weiterer Programmverlauf
  .
900 DATA NAME,VORNAME,STRASSE,WOHNORT,XXX
910 DATA ARTIKEL,ARTIKELNR.,PREIS,LAGERORT,SOLL,IST,XXX
920 DATA TITEL,AUTOR,ISBN-NR.,ERSCHIENEN,GEKAUFT AM,XXX
```

Diese ist sicherlich die elegantere der zwei möglichen Methoden, bindet den Anwender in gewisser Weise jedoch immer noch an ein festgelegtes Schema. Da wir uns in keiner Weise binden wollen, verwenden wir in unserer weiteren Programmierung die zweite Möglichkeit, die jedoch mehr Speicherplatz beansprucht.

2. Ein sehr flexibles Datenverwaltungsprogramm kann man mit dem folgenden Verfahren aufbauen. Hier werden alle Abfragekriterien vor der eigentlichen Dateneingabe durch den jeweiligen Anwender mittels INPUT-Befehl definiert. Nach diesen Kriterien wird nun die Dateneingabe gesteuert. Man kann das Programm in seiner Vielseitigkeit noch verbessern, wenn vor der Definition der Abfragekriterien ein INPUT-Befehl eingebaut wird, über den die Anzahl der gewünschten Datenelemente eingegeben werden kann. In unserem Beispielprogramm wollen wir Ihnen die Methode etwas anschaulicher darbieten:

```
10 REM ***************************************************
20 REM *** BEISPIEL FUER DIE EINGABE VON ABFRAGEKRI-  ***
30 REM ***                TERIEN                     ***
40 REM ***************************************************
50 CALL CLEAR
60 REM ** DIMENSIONIERUNG DER BENOETIGTEN ARRAY'S      **
70 REM ** FUER 50 DATENSAETZE MIT JEWEILS 10 ELEMENTEN **
80 DIM DA$(50,10),BEZ$(10)
90 OPTION BASE 1
100 INPUT 'ANZAHL DER ELEMENTE:':B
110 PRINT
120 CALL CLEAR
130 IF B>10 THEN 100
140 IF B<0 THEN 100
150 PRINT 'EINGABE DER ELEMENTE:'
160 PRINT :
170 FOR I=1 TO B
180 PRINT 'ELEMENT';I;
190 INPUT BEZ$(I)
200 NEXT I
210 CALL CLEAR
220 REM ** EINGABE DER DATEN **
```

```
230 Z=1
240 FOR I=1 TO B
250 PRINT BEZ$(I);
260 INPUT DA$(Z,I)
270 IF DA$(Z,I)='STOP' THEN 330
280 NEXT I
290 Z=Z+1
300 IF Z>50 THEN 330
310 CALL CLEAR
320 GOTO 240
330 Z=Z-1
340 CALL CLEAR
350 PRINT 'ZUM AUFLISTEN DER EINGE-'::
360 PRINT 'GEBENEN DATEN:'::::
370 PRINT 'DRUECKE EINE TASTE!'
380 CALL KEY(0,K,S)
390 IF S=0 THEN 380
400 CALL CLEAR
410 FOR I=1 TO Z
420 FOR J=1 TO B
430 PRINT BEZ$(J);':';DA$(I,J)
440 NEXT J
450 PRINT ::'DRUECKE EINE TASTE!'
460 CALL KEY(0,K,S)
470 IF S=0 THEN 460
480 CALL CLEAR
490 NEXT I
500 END
```

Sie können jetzt sehr leicht selbst abwägen, welches der beiden vorgestellten Verfahren in Ihren Programmen Anwendung findet. Vielleicht haben Sie sogar schon längst eine eigene Methode entwickelt, um universelle Datenverwaltungsprogramme zu erstellen !? Dem Anfänger haben wir hier zweifellos ein nützliches Instrument an die Hand gegeben.

4.8 Ein Programm für die Datenverwaltung

Wir wollen Ihnen nicht nur Beispiele präsentieren, sondern auch ein komplettes Datenverwaltungsprogramm bieten! Dies soll hier geschehen. Nach einer ausführlichen Bedienungsanleitung für das Programm, folgt eine intensive Beschreibung des im Anschluß abgedruckten Listings.

Bis zu 200 Datensätze a 7 Informationen können Sie mit der folgenden Datenverwaltung erfassen. Das nachstehend aufgelistete Programm ist recht komfortabel, wird jedoch bei manchem den einen oder anderen Wunsch offen lassen. Eine Datenverwaltung für zu Hause oder den kleinen Verein ist jedoch voll zufriedenstellend möglich.

Nach dem Programmstart mit RUN wird Ihnen auf dem Bildschirm ein Menu generiert, das eine ganze Reihe nützlicher Funktionen zur Auswahl bietet:

```
1-> DATEI ERSTELLEN
2-> DATEN EINLESEN
3-> DATEN ABSPEICHERN
4-> DATENBEHANDLUNG
5-> PROGRAMMENDE
```

Wenn Sie nun die '1' eingeben, werden Sie zur eigentlichen Dateneingabe geführt. Hier müssen Sie zuerst den Namen für die Datei eingeben und werden dann nach der Anzahl der Datenelemente gefragt. Beispiel:

```
DATEINAME? ADRESSDATEI
ANZAHL DER ELEMENTE? 7
```

Wir haben nun dem Programm mitgeteilt, daß eine Adressenverwaltung aufgebaut werden soll, die je Datensatz 7 Informationen enthält, Jetzt müssen die Namen eingegeben werden, mit denen jedes einzelne Element bezeichnet werden soll (Abfragekriterien). Beispiel:

ELEMENT 1? NAME
ELEMENT 2? VORNAME
ELEMENT 3? STRASSE
ELEMENT 4? WOHNORT
ELEMENT 5? BERUF
ELEMENT 6? TELEFON
ELEMENT 7? BEMERKUNG

Haben Sie all diese Eingaben abgeschloßen, wird die Eingabe der Daten (Adressen in unserem Beispiel) eingeleitet. Wenn Sie den Eingabemodus verlassen wollen, geben Sie bei dem ersten Element einfach 'XXX' ein, und das Hauptwahlmenu erscheint wieder auf dem Bildschirm.

Mit den Kennziffern '2' und '3' können Sie Ihre erfassten Daten auf Kassettenrekorder abspeichern bzw. fertige Dateien von Band einlesen. Wenn Sie eine Datei abspeichern, werden grundsätzlich alle Elementbezeichnungen mit auf Cassette gespeichert: Lesen Sie also eine fertige Datei in das Programm ein, so brauchen die Abfragekriterien n i c h t neu definiert zu werden.

Kennziffer '4' führt Sie zur Datenbehandlung. Hier wird zunächst ein Untermenu erzeugt, das Ihnen folgende Wahlmöglichkeiten bietet:

1-> DATEN AUFLISTEN
2-> DATEN SUCHEN
3-> DATEN AENDERN
4-> RUECKSPRUNG

Unter '1' werden immer die Informationen des 1. Elementes eines jeden Datensatzes aufgelistet. Dadurch ist es Ihnen möglich, alle erfassten Daten schnell zu überblicken. Es werden immer nur soviele Datenelemente auf dem Bildschirm ausgegeben, daß ein 'scrollen' vermieden wird. Es ist nun möglich, durch Eingabe eines Kennbuchstabens die Datei Bildschirmseite für Bildschirmseite durchzublättern.
Folgende Möglichkeiten stehen zur Verfügung:

V = Vorwärts blättern
R = Rückwärts blättern
O = Rücksprung zum Untermenu

Wahl '2' ermöglicht das Suchen bestimmter Datensätze nach einem, von Ihnen bestimmten Kriterium. Sie müssen nun nach folgendem Beispiel verfahren:

DATEN SUCHEN:

1- NAME
2- VORNAME
3- STRASSE
4- WOHNORT
5- BERUF
6- TELEFON
7- BEMERKUNGEN

BITTE KENNZIFFER EINGEBEN!

Nehmen wir an, es sei eine Adressdatei in unserem Programm gespeichert, dann kann, wenn die Kennziffer '2' gedrückt wurde, die oben aufgelistete Tabelle auf dem Bildschirm erscheinen. Aus dem gesuchten Datensatz ist nur die Telefonnummer bekannt. Da unter der Nummer '6' das Kriterium 'TELEFON' aufgeführt ist, geben wir jetzt diese Kennziffer ein

und werden sofort nach der Telefonnummer gefragt, die jetzt einzugeben ist:

TELEFON? 06355/16653

Nach kurzer Zeit wird der entsprechende Datensatz - vorausgesetzt er ist gespeichert - komplett auf dem Bildschirm ausgegeben.

Kennziffer '3' ermöglicht es, Datensätze zu ändern. Nach Eingabe der Datensatznummer (kann man unter '1' beim Auflisten ermitteln), wird der gewählte Datensatz Element für Element auf dem Bildschirm angezeigt. Jetzt kann man entweder das angezeigte Element ändern, indem man die neue Information eingibt, oder man drückt einfach 'ENTER' und beläßt die alte Information im Speicher.

Drückt man die '4', so wird ins Hauptmenu zurückgesprungen.

Damit Sie das Programm richtig verstehen, wird jetzt Zeile für Zeile des Listings abgehandelt:

10 Bildschirm löschen.

20 Programmtitel in der Bildschirmmitte ausgeben.

30 Dimensionierung der Variablen DAT$(X,Y), die zur Datenspeicherung dient. Durch Ändern der Klammerwerte können Sie die Kapazität des Programmes neu festlegen. In wieweit Sie Ihr Programm mit Daten auslasten können, errechnen Sie bitte nach dem in Kapitel 2.2 vorgestellten Verfahren.

40 - 95 Erzeugen des Hauptmenu's.

100-105 Warten bis eine beliebige Taste gedrückt wird.

110-111 Überprüfung, ob die gedrückte Taste im gültigen Bereich liegt.

120 Verzweigung in die angewählten Programmodule.

130 Programmende.

140 Bildschirm löschen.

145-200 In diesen Zeilen wird der Datei ein Name gegeben, und das Datum des Eingabetages wird eingetippt. Anschließend muß dem Programm mitgeteilt werden, aus wievielen Datenelementen ein Datensatz besteht. Zeile 200 löscht wieder den Bildschirm.

201-230 Hinweis, daß beim Ende der Eingabe das Wort 'STOP' eingegeben werden muß.

240-250 Warten auf das Drücken der Leertaste.

260 Das Kontrollregister wird auf seinen Anfangswert gesetzt.

270-310 Eingabe der Elementbezeichnungen, die als Abfragekriterien dienen.

380-500 Hier findet die eigentliche Dateneingabe statt. In der Zeile 440 wird überprüft, ob das Wort 'STOP' eingegeben wurde.

510-530 Ende der Dateneingabe.

540 - 640 Routine zum Einlesen von Daten vom Kassettenrekorder.

650 - 770 Routine zum Abspeichern der eingegebenen Daten auf Band.

780 - 781 Bildschirm löschen und Aufruf der Subroutine, die den Namen, das Eingabedatum und die Anzahl der gespeicherten Datensätze auf dem Bildschirm ausgibt.

790 - 850 Erzeugen des Untermenu's für die Datenbehandlung

860 - 871 Warten auf das Drücken einer Taste und Abfrage, ob die gedrückte Taste im gültigen Bereich liegt.

880 Verzweigen zu dem angewählten Programmodul.

890 -1025 Hier werden die Anfangswerte der auf dem Bildschirm aufzulistenden Datensätze gesetzt und die Ausgaberoutine (1050) aufgerufen.

1030 Verzweigen in eine Blindzeile (Blindzeilen werden zur Programmkennzeichnung verwendet und haben auf den Programmfluß keine Auswirkungen).

1040 Rücksprung zum Untermenu.

1050-1115 Ausgaberoutine zum Auflisten der Datensätze.

1130-1135 Warten auf das Drücken einer Taste.

1140 Wenn die Taste mit ASCII-Code 86 gedrückt wurde, springt der Programmzeiger zum Ausgangspunkt zu-

rück. Ansonsten wird nach Zeile 1150 gesprungen.

1150-1170 Überprüfung, welche der Steuertasten gedrückt wurden und entsprechende Verzweigung in die Zeilen 1290 bzw. 780 (Untermenu).

1171-1245 Ausgabe eines gewünschten Datensatzes, nachdem in Zeile 1171 die Datensatznummer eingegeben wurde.

1250-1270 Warten auf das Drücken der Leertaste.

1290-1380 Überprüfung der momentan aufgelisteten Datensatznummern für das Rückwärtsblättern.

1390-1512 Routine zum Suchen von Daten. Nach der Eingabe von Elementnummer (1415) und Bezeichnung (1425) wird in Zeile 1480 der Datensatz gesucht. Wenn nach Durchlauf der gesamten Datei der gesuchte Datensatz nicht gefunden wird, erfolgt in Zeile 1505 eine Fehlermeldung. Ist der Begriff gefunden worden, verzweigt das Programm nach 1520 um dort den kompletten Datensatz auszugeben.

1520-1610 Ausgabe des gesuchten Datensatzes.

1610-1690 Unterprogramm zur Ausgabe von Namen, Erstellungsdatum und Anzahl der gespeicherten Datensätze.

1700 Blindzeile (kann ohne weiteres aus dem Programm herausgelassen werden).

1710-1870 Ändern von Datenelementen. In Zeile 1730 wird die Nummer des zu ändernden Datensatzes einge-

geben, der dann Element für Element auf dem Bildschirm ausgegeben wird (1790). Zeile 1796 überprüft, ob nur die 'ENTER'-Taste gedrückt wurde.

```
10 CALL CLEAR
20 PRINT TAB(6);"DATENVERWALTUNG"
30 DIM DAT$(200,7)
40 PRINT
45 PRINT
48 PRINT "1-> DATEI ERSTELLEN"
50 PRINT
55 PRINT "2-> DATEN EINLESEN"
60 PRINT
65 PRINT "3-> DATEN ABSPEICHERN"
70 PRINT
75 PRINT "4-> DATENBEHANDLUNG"
80 PRINT
85 PRINT "5-> PROGRAMMENDE"
87 PRINT
90 PRINT
95 PRINT "BITTE KENNZIFFER EINGEBEN"
100 CALL KEY(0,K,S)
105 IF S=0 THEN 100
110 IF K-48<1 THEN 100
111 IF K-48>5 THEN 100
120 ON K-48 GOTO 140,540,650,780,130
130 END
140 CALL CLEAR
145 PRINT TAB(6);"DATEIERSTELLUNG"
146 PRINT
147 PRINT
148 PRINT
150 INPUT "DATEINAME:":NAME$
155 PRINT
160 INPUT "DATUM     :":DA$
165 PRINT
170 INPUT "ANZAHL DATENELEMENTE:":ANZ
200 CALL CLEAR
201 PRINT
202 PRINT
203 PRINT
210 PRINT "BEI ENDE DER DATENEINGABE"
215 PRINT
220 PRINT "==> 'STOP' EINGEBEN!"
225 PRINT
227 PRINT
230 PRINT "MIT LEERTASTE WEITER"
240 CALL KEY(0,K,S)
245 IF S=0 THEN 240
250 IF K<>32 THEN 240
260 KONTR=3
270 CALL CLEAR
275 PRINT "ELEMENTBEZEICHNUNG:"
280 FOR Z=1 TO ANZ
290 KONTR=KONTR+2
300 PRINT
305 PRINT "ELEMENT";Z;":";
308 INPUT A$(Z)
310 NEXT Z
380 KONTR=3
380 FOR Z=1 TO 200
390 CALL CLEAR
```

```
395 PRINT "DATENSATZ";Z
396 PRINT
397 PRINT
400 SATZ=SATZ+1
410 FOR X=1 TO ANZ
420 KONTR=KONTR+2
430 PRINT A$(X);":";
435 INPUT DAT$(Z,X)
440 IF DAT$(Z,X)="STOP" THEN 441 ELSE 450
441 DAT$(Z,X)=" "
442 SATZ=SATZ-1
443 GOTO 10
450 REM
500 NEXT X
510 KONTR=3
520 NEXT Z
530 GOTO 10
540 CALL CLEAR
550 OPEN #1:"CS1",SEQUENTIAL,INTERNAL,INPUT,FIXED
560 INPUT #1:NAME$,DA$,ANZ,SATZ
570 FOR Z=1 TO 9
571 INPUT #1:A$(Z)
572 NEXT Z
580 FOR Z=1 TO SATZ
590 FOR X=1 TO ANZ
600 INPUT #1:DAT$(Z,X)
610 NEXT X
620 NEXT Z
630 CLOSE #1
640 GOTO 10
650 CALL CLEAR
660 OPEN #1:"CS1",SEQUENTIAL,INTERNAL,OUTPUT,FIXED
670 PRINT #1: NAME$,DA$,ANZ,SATZ
680 FOR Z=1 TO 9
690 PRINT #1:A$(Z)
700 NEXT Z
710 FOR Z=1 TO SATZ
720 FOR X=1 TO ANZ
730 PRINT #1:DAT$(Z,X)
740 NEXT X
750 NEXT Z
760 CLOSE #1
770 GOTO 10
780 CALL CLEAR
781 GOSUB 1620
790 PRINT "BITTE WAEHLEN SIE!"
791 PRINT
792 PRINT
793 PRINT
800 PRINT "1-> DATEN AUFLISTEN"
805 PRINT
810 PRINT "2-> DATEN SUCHEN"
815 PRINT
820 PRINT "3-> DATEN AENDERN"
830 PRINT
840 PRINT "4-> RUECKSPRUNG"
845 PRINT
846 PRINT
850 PRINT "BITTE KENNZIFFER EINGEBEN"
860 CALL KEY(0,K,S)
```

```
865 IF S=0 THEN 860
870 IF K-48<1 THEN 860
871 IF K-48>4 THEN 860
880 ON K-48 GOTO 890,1390,1720,10
890 CALL CLEAR
900 PRINT "DATENLISTE:"
910 ANF=1
920 GOSUB 1050
930 ANF=21
940 GOSUB 1050
950 ANF=41
955 GOSUB 1050
960 ANF=61
965 GOSUB 1050
970 ANF=81
975 GOSUB 1050
980 ANF=101
985 GOSUB 1050
990 ANF=121
995 GOSUB 1050
1000 ANF=141
1005 GOSUB 1050
1010 ANF=161
1015 GOSUB 1050
1020 ANF=181
1025 GOSUB 1050
1030 GOSUB 1700
1040 GOTO 780
1050 ZEILE=1
1060 FOR Z=ANF TO ANF+19
1070 ZEILE=ZEILE+1
1080 PRINT "SATZ:";Z;TAB(10);DAT$(Z,1)
1110 NEXT Z
1111 M$="BITTE NUMMER/CODE EING.:"
1112 FOR I=1 TO LEN(M$)
1113 M1=ASC(SEG$(M$,I,1))
1114 CALL HCHAR(24,I,M1)
1115 NEXT I
1130 CALL KEY(0,K,S)
1135 IF S=0 THEN 1130
1140 IF K=86 THEN 1141 ELSE 1150
1141 RETURN
1150 IF K=82 THEN 1290
1160 IF K=48 THEN 780
1170 CALL CLEAR
1171 INPUT "SATZNUMMER:":WAHL
1180 CALL CLEAR
1190 KONTR=3
1200 FOR Z=1 TO ANZ
1210 KONTR=KONTR+2
1220 PRINT A$(Z);":";DAT$(WAHL,Z)
1221 PRINT
1222 PRINT
1230 NEXT Z
1240 PRINT"******************************"
1245 PRINT "MIT LEERTASTE WEITER"
1250 CALL KEY(0,K,S)
1255 IF S=0 THEN 1250
1260 IF K<>32 THEN 1250
1270 CALL CLEAR
```

```
1280 GOTO 1050
1290 IF ANF<=20 THEN 910
1300 IF ANF<=40 THEN 910
1310 IF ANF<=60 THEN 930
1320 IF ANF<=80 THEN 950
1330 IF ANF<=100 THEN 960
1340 IF ANF<=120 THEN 970
1350 IF ANF<=140 THEN 980
1360 IF ANF<=160 THEN 990
1370 IF ANF<=180 THEN 1000
1380 IF ANF<=120 THEN 1010
1390 CALL CLEAR
1400 PRINT TAB(8);"DATEN SUCHEN"
1410 PRINT
1411 PRINT
1415 INPUT "ELEMENT-NR.:":EN
1420 PRINT
1425 INPUT "BEZEICHNUNG:":BEZ$
1450 CALL CLEAR
1460 PRINT TAB(8);"BITTE WARTEN!"
1461 PRINT
1462 PRINT
1463 PRINT
1464 PRINT
1465 PRINT
1470 FOR Z=1 TO 200
1480 IF BEZ$=DAT$(Z,EN) THEN 1520
1490 NEXT Z
1500 CALL CLEAR
1505 PRINT "DATEN NICHT GEFUNDEN!!!!"
1510 FOR X=1 TO 3000
1511 NEXT X
1512 GOTO 780
1520 KONTR=3
1530 CALL CLEAR
1540 FOR X=1 TO ANZ
1550 KONTR=KONTR+2
1560 PRINT
1565 PRINT A$(X);":";DAT$(Z,X)
1570 NEXT X
1580 PRINT
1581 PRINT
1585 PRINT "MIT LEERTASTE WEITER"
1590 CALL KEY(0,K,S)
1595 IF S=0 THEN 1590
1600 IF K<>32 THEN 1590
1610 GOTO 780
1620 PRINT TAB(5);NAME$
1630 PRINT
1631 PRINT
1632 PRINT TAB(5);DA$
1640 PRINT
1641 PRINT
1645 PRINT TAB(5);SATZ;"DATENSAETZE"
1650 PRINT
1651 PRINT
1652 PRINT
1655 PRINT "MIT LEERTASE WEITER"
1660 CALL KEY(0,K,S)
1665 IF S=0 THEN 1660
```

```
1670 IF K<>32 THEN 1660
1680 CALL CLEAR
1690 RETURN
1700 RETURN
1710 REM
1720 CALL CLEAR
1725 PRINT "DATEN AENDERN:"
1725 PRINT
1726 PRINT
1727 PRINT
1730 INPUT "WELCHER DATENSATZ?:":DS
1750 CALL CLEAR
1760 KONTR=3
1770 FOR Z=1 TO ANZ
1780 PRINT
1785 PRINT
1790 PRINT A$(Z);":";
1795 INPUT DAT$(DS,Z):V$
1796 IF V$="" THEN 1820
1800 DAT$(DS,Z)=V$
1820 NEXT Z
1870 GOTO 780
```

KAPITEL 5: TIP'S UND TRICK'S

In diesem Kapitel sollen nützliche Routinen und kleine Programmierkniffe vorgestellt werden. Der geübte Programmierer wird diese kleinen Hilfsroutinen in abgewandelter Form sicherlich schon kennen. Für Ihn ist dieses Kapitel wahrscheinlich relativ uninteressant, doch wird der Einsteiger hier an Programmierweisen herangeführt, die er sich erst nach langer Zeit hätte selbst erarbeiten können.

Die nachfolgend aufgeführten Routinen stützchen sich im wesentlichen auf TI-Basic, sind in einigen Fällen aber auch auf EXTENDED BASIC übertragbar.

Um Ihnen das Umschreiben fremder Programme zu erleichtern, wurden noch einige Befehle des Microsoftbasic in dieses Kapitel mit aufgenommen um zu zeigen, wie man diese Befehle am einfachsten umschreibt. Einige dieser Befehle haben wir im Kapitel 2 schon einmal erklärt und gehen hier etwas ausführlicher darauf ein.

5.1 Realisierung der logischen Verknüpfungsglieder AND (UND), OR (ODER) und NOT (NICHT)

Diese Verknüpfungsglieder sind in Microsoftbasic als Statements implementiert. Mit einer logischen Verknüpfung wird eine Verzweigung unter bestimmten Bedingungen durchgeführt. In der Regel werden zwei oder mehrere IF.....-Bedingungen miteinander verknüpft.

In TI-BASIC sind diese Befehle nicht verfügbar, können aber leicht simuliert werden, wie im Folgenden beschrieben wird.

5.1.1 Die UND-Funktion (AND)

Eine Verzweigung wird hier vorgenommen, wenn die eine U N D die andere Bedingung erfüllt ist.

Beispiel: Verzweigung nach 200, wenn A größer als 6 und B kleiner als 11 ist. Hier wird die Programmzeile in Microsoftbasic folgendermaßen aussehen:

```
100 IF A>6 AND B<11 THEN 200
```

In TI-BASIC muß man diese Zeile wie folgt programmieren:

```
100 IF A>6 THEN 101 ELSE 110
101 IF B<11 THEN 200
110 END
```

Wenn die Bedingung in Zeile 100 mit 'JA' beantwortet werden kann, erfolgt eine Verzweigung nach 101 um dort die zweite Bedingung zu testen. Nur wenn beide Bedingungen mit 'JA' beantwortet werden können, erfolgt eine Verzweigung nach 200 andernfalls wird das Programm beendet oder führt andere Statements durch. Wie an anderer Stelle schon beschrieben, können Sie auf diese Art und Weise beliebig viele Vergleiche mit UND (AND) verknüpfen.

5.1.2 Die ODER-Funktion (OR)

Diese Funktion wird dann eine Verzweigung herbeiführen, wenn bei einem Vergleich die eine O D E R die andere Bedingung erfüllt ist.

Beispiel: Verzweigung nach 1000, wenn A gleich A1 ODER C kleiner als C1 ist. Hier die Zeile in Microsoft-basic:

```
100 IF A=A1 OR C<C1 THEN 1000
```

und nun die Realisierung in TI-BASIC:

```
100 IF A=A1 THEN 1000
200 IF C<C1 THEN 1000
```

Wie Sie sehen, ist das Umsetzen einer ODER-Verknüpfung sehr einfach. Wird in Zeile 100 die Bedingung mit 'JA' beantwortet, so wird das Programm nach 1000 verzweigen. Andernfalls wird die Zeile 200 abgearbeitet um im 'JA'-Fall nach 1000 zu springen. Werden beide Bedingungen mit 'NEIN' beantwortet, werden die, nach 200 folgenden Anweisungen durchgeführt.

5.1.3 Die NICHT-Funktion (NOT)

Dieser Vergleichsbefehl wird eine Verzweigung herbeiführen, wenn eine Bedingung N I C H T zutrifft.

Beispiel: Verzweigung nach 500, wenn der Inhalt der Variablen A NICHT gleich 25 ist. In Microsoftbasic:

```
100 IF NOT A=25 THEN 500
```

In TI-BASIC:

```
100 IF A<>25 THEN 500
```

Wenn A ungleich 25 ist, erfolgt ein Sprung nach 500. Wie Sie oben sehen können, ist die NOT-Funktion für eine Programmierung nucht unbedingt von Vorteil.

5.1.4 Die Verknüpfung von UND und ODER

Eine sehr leistungsfähige Programmiertechnik ist das Verknüpfen dieser beiden Funktionen. Auf diese Art kann man (fast) jedes Prolem formulieren und lösen. Das folgende Beispiel gibt einen kleinen Einblick in die hervorragenden Möglichkeiten dieser Technik. Mit dieser Übung werden Sie sehr schnell die Vorteile die diese Technik bietet zu schätzen wissen.

Beispiel: Eine Verzweigung soll dann stattfinden, wenn die Variable A gleich X ist und die Variable M klei-

ner F oder die Variable c gleich 75 ist. Wenn Sie den Aufgabentext aufmerksam gelesen haben, wissen Sie bereits, daß zur Lösung zwei Verknüpfungen benötigt werden. Dem logischen Baustein UND muß, wie die Schaltalgebra festlegt, der Vorrang gegeben werden.

Die Lösung in Microsoftbasic:

```
100 IF A=X AND (M<F OR C=75) THEN 1000
```

Die etwas kompliziertere Lösung in TI-BASIC:

```
100 IF A=X THEN 101 ELSE 110
101 IF M<F THEN 1000
102 IF C=75 THEN 1000
110 END (oder weitere Statements)
```

Um eine Verzweigung herbeizuführen muß in jedem Falle die Bedingung in Zeile 100 erfüllt sein, sonst wird das Programm ab Zeile 110 fortfahren. Wenn diese Bedingung in Zeile 110 erfüllt ist, wird überprüft, ob eine der beiden Bedingungen in den Zeilen 101 und 102 mit 'JA' beantwortet werden können. Ist dies der Fall, so verzweigt das Progamm nach 1000.

5.2 Die Zahl PI

An dieser Stelle soll auch die Berechnung der Zahl PI, die ja bei TI-BASIC nicht implementiert ist, gezeigt werden. Sollten Sie in einem Programm mehrmals diese Zahl zur Berechnung verschiedener Funktionen benutzen, kommt Ihnen diese Berechnungsformel sicherlich zugute, da Sie die Zahl PI nur einmal zuzuweisen brauchen. Geben Sie einmal im Direktmodus folgende Zeile ein:

```
PI=(4*ATN(1))
PRINT PI
```

Sie sehen, daß nun die genaue Zahl PI berechnet wurde.

Anmerkung: Wenn Sie über das EXTENDED-BASIC-Modul verfügen, haben Sie diese Zahl sofort nach Aufruf der Variablen >PI< zur Verfügung.

5.3 Die Microsoft-Befehle LEFT$, MID$ und RIGHT$

Wenn Sie auf das große Angebot an Programmen in Microsoftbasic nicht verzichten wollen, müssen Sie die, dort häufig verwendeten String-Funktionen MID$, LEFT$ und RIGHT$ auf TI-BASIC umsetzen. Bei den Funktionen LEFT$ und MID$ werden Sie keine Schwierigkeiten haben, da diese Befehle dem TI-BASIC-Befehl SEG$ weitgehend entsprechen. Lediglich bei RIGHT$ wird der Programmieraufwand etwas umständlich sein. Im Folgenden soll an kurzen Beispielen gezeigt werden, wie man diese Funktionen für TI-BASIC umsetzt (siehe auch Kap. 2):

5.3.1 LEFT$ (A$,L)

Diese Funktion wird benutzt, wenn man eine bestimmte Zeichenlänge (L) eines Strings (A$), von links aus gesehen, absondern will.

Beispiel: Von dem String 'DATA BECKER' sollen die ersten 4 Zeichen in der Stringvariablen B$ abgespeichert werden. In Microsoftbasic sieht dieses folgendermaßen aus:

```
100 A$='DATA BECKER'
110 B$=LEFT$(A$,4)
120 PRINT B$
```

IN TI-BASIC, wo wir als einzige Stringfunktion SEG$ haben, muß das Problem so programmiert werden:

```
100 A$='DATA BECKER'
110 B$=SEG$(A$,1,4)
120 PRINT B$
```

In beiden Fällen wird nun in der Variablen B$ die Information 'DATA' stehen.

5.3.2 MID$ (A$,A,L)

Mit dieser Funktion ist es möglich, einen Teilstring mitten aus einem String zu bekommen. A$ stellt den String dar, aus dem ein Teilstring gewonnen werden soll. A ist die Position, an der ein String mit der Länge L beginnt.

Beispiel: Aus dem String 'DATA BECKER' soll ab der 3. Stelle ein Teilstring mit einer Länge von 3 Zeichen der Variablen B$ zugewiesen werden. Zuerst die Microsoft-Version:

```
100 A$='DATA BECKER'
110 B$=MID$(A$,3,3)
120 PRINT B$
```

Diese Zeilen sehen in TI-BASIC fast genauso aus:

```
100 A$='DATA BECKER'
110 B$=SEG$(A$,3,3)
120 PRINT B$
```

Auch hier wird in beiden Fällen der Variablen B$ die gleiche Zeichenfolge - 'TA ' - zugewiesen.

5.3.3 RIGHT$ (A$,L)

Bei dieser Funktion bekommt man eine Zeichenkette bestimmter Länge (L) aus einem String (A$) von rechts aus gesehen. Hier ist die Programmierung, im Vergleich zu den anderen beiden Funktionen, eher kompliziert.

Beispiel: Aus unserem String 'DATA BECKER' sollen der Stringvariablen B$ die letzten 4 Zeichen zugewiesen werden. In Microsoftbasic ist dies sehr einfach:

```
100 A$='DATA BECKER'
110 B$=RIGHT$(A$,4)
120 PRINT B$
```

Will man nun in TI-BASIC programmieren, muß dies so aussehen:

```
100 A$='DATA BECKER'
110 A=LEN(A$)
120 IF A-4 <=0 THEN 130 ELSE 140
130 B$=A$
135 GOTO 150
140 B$=SEG$(A$,A-4+1,4)
150 PRINT B$
```

In der Zeile 110 wird zuerst festgestellt, wie lang der String aus A$ ist. Danach wird in Zeile 120 getestet, ob die Länge des Strings ausreichend ist, um die benötigte Zeichenzahl zu bekommen. Ist die Länge nicht ausreichend, wird der Variablen B$ die komplette Information aus A$ zugewiesen (130) und ausgegeben (150). In der Zeile 140 wird nun als

Anfangswert der Länge des Strings (A) die Zahl 4 subtrahiert und 1 hinzuaddiert. Der Rest ist sehr einfach, wie Sie oben sehen können. Die Variable B$ wird in beiden Beispielen den Teilstring 'CKER' enthalten.

5.4 Ausdrucken an beliebiger Bildschirmposition mit CALL HCHAR

Leider ist im TI-BASIC kein Befehl implementiert, der einen direkten Ausdruck an einer beliebigen Bildschirmposition, analog des EXTENDED BASIC Befehls DISPLAY AT, erlaubt. Es ist jedoch möglich, mit Hilfe der Routine HCHAR diesen Befehl leicht zu simulieren.

Beispiel: Es soll in der 5. Zeile, ab Spalte 8, der Begriff 'DATA BECKER' ausgegeben werden:

```
100 A$='DATA BECKER'
110 A=LEN(A$)
120 FOR I=1 TO A
130 B$=SEG$(A$,I,1)
140 B=ASC(B$)
150 CALL HCHAR(5,7+I,B)
160 NEXT I
170 GOTO 170
```

Wenn Sie dieses Programm einmal ausprobieren, werden Sie feststellen, daß der Ausdruck wie oben beschrieben erfolgt.

Nach der Zuweisung des Strings (100) wird dessen Länge bestimmt (110). Mit dem Wert (A) wird eine Zählschleife angesteuert, die zur Decodierung der einzelnen Zeichen und der

Ausgabe dient. Nachdem jedes einzelne Zeichen abgesondert wurde (130) und in den ASCII-Code umgewandelt ist (140), erfolgt die Ausgabe per CALL HCHAR zeichenweise ab der spezifizierten Bildschirmposition.

5.5 Datenspeicherung auf Cassette

Sicherlich haben Sie auch schon festgestellt, daß der TI 99/4A bei der Datenspeicherung auf Cassette relativ langsam arbeitet. Besonders wenn mehrere Datensätze über eine Schleife abgespeichert werden sollen, dauert das Überspielen unerträglich lange. Die Ursache liegt daran, daß das System bei jedem Erreichen des NEXT-Statements kurz anhält (2-3 Sekunden), um dann den nächsten PRINT # oder INPUT # Befehl in der Schleife durchzuführen. Der TI 99/4A hat auch keinen ausreichenden Bandpuffer, wie er bei vergleichbaren Systemen üblich ist (im Bandpuffer werden vor der Datenübertragung Informationen 'gesammelt', die dann in einem Vorgang auf das Band geschrieben/von Band gelesen werden). Auch dieses ist mit ein Grund für die langsame Arbeitsweise.

Man kann sehr wenig tun, um die Datenspeicherung zu beschleunigen. Lediglich bei der Datenspeicherung über eine Schleifensteuerung kann man durch einen kleinen Kniff den Datentransfer unter Umständen erheblich forcieren.

Beispiel: Will man 100 Datensätze abspeichern, wird man in der Regel folgendermaßen programmieren:

```
100 OPEN #1:'CS1',SEQUENTIAL,INTERNAL,OUTPUT
110 FOR I=1 TO 100
120 PRINT #1:A$(I)
```

```
130 NEXT I
140 CLOSE #1
```

Bei dieser Routine wird das Bandgerät 99 mal zwischen den einzelnen Datenübertragungen anhalten. Daraus resultiert eine Wartezeit von zusammen ungefähr 120 Sekunden. Nimmt man jetzt noch die Zeitdauer, die der Rechner braucht um die Daten zu speichern hinzu (diese Zeitdauer kann je nach Datenmenge pro PRINT #-Befehl variieren), kommt man leicht auf ca. 250 Sekunden (über 4 Minuten mindestens).

Im nächsten Beispiel wollen zeigen, wie man die Anzahl der Zwischenstops reduzieren und somit die Datenübertragung wesentlich beschleunigen kann.

Beispiel:

```
100 OPEN #1:'CS1',SEQUENTIAL,INTERNAL,OUTPUT
110 FOR I=1 TO 91 STEP 10
120 PRINT #1:A$(I),A$(I+1),A$(I+2),A$(I+3),A$(I+
4),A$(I+5),A$(I+6),A$(I+7),A$(I+8),A$(I+9)
130 NEXT I
140 CLOSE #1
```

Durch oben stehende Routine erreichen wir, daß der Rechner zwischen den einzelnen Speichervorgängen nur 9 mal anhält, so daß wir mit der kompletten Datenspeicherung nach ca. 90 Sekunden fertig sein werden. Wie kommt das ?

In Zeile 110 wird die Schleife zur Datenspeicherung angesteuert, die nach jedem Durchlauf um 10 erhöht wird. Nun wird in der Schleife durch den PRINT #-Befehl nicht eine, sondern zehn Informationen abgespeichert. Genau darin liegt die jetzt schnellere Arbeitsweise dieser Routine, die so nur 10 mal durchlaufen wird.
Der etwas größere Programmieraufwand wird sich in den meisten Fällen sicherlich lohnen.

5.6 Speicherplatzeinsparung bei der Datenverarbeitung

Die 16 K-Byte Speicherkapazität des TI 99/4A läßt den Anwender - vor allem bei der Verarbeitung größerer Datenmengen - schnell an die Grenzen der Grundausstattung stoßen. Nun ist man entweder auf die Anschaffung einer, nicht gerade preiswerten Speicherung angewiesen oder man schafft es, durch geschickte Ausnutzung der Möglichkeiten mit dem zur Verfügung stehenden Speicherplatz auszukommen. Im folgenden wird eine solche Methode aufgezeigt, mit der Sie in der Lage sind, eine Reihe verschiedener Informationen in einer einzigen Variablen abzuspeichern. Der Clou dabei ist, daß alle Informationen einzeln verarbeitet werden können.

Dieses Verfahren benötigt eine umfangreiche Decodierroutine, die den Speicherplatz für das Programm selbst um ca. 0.5 K vergrößert. Deshalb können wir diese Methode nur dort empfehlen, wo auch wirklich größere Datenmengen abgespeichert werden sollen (> 300 Datensätze).

Unser Beispiel wird so aufgebaut sein, daß jede Zeile einzeln abgehandelt wird. Dies führt zwar unter Umständen zu einer gewissen Unübersichtlichkeit, ist aber wegen der komplizierten Programmiertechnik unvermeidbar.

Beispiel:

Es sollen durch einen INPUT-Befehl die Daten einer Lagerkarte eingegeben werden. Auf dieser Karte sind 25 Schrauben 8x4 mit der Bestellnummer 512-11-8064 eingetragen. Als weitere Informationen sind noch die Sollmenge von 50 Stück und die Nummer des Lagerortes angegeben (A511). Im Endeffekt sollen 300 dieser Karten verarbeitet werden.

Nun zum Erstellen des Programmes:

Als erstes müssen wir definieren, wieviele Datensätze erfasst werden sollen. Dies geschieht über die Dimensionierung eines Datenfeldes.

```
10 DIM A$(300)
```

Da ja mehrere Informationen unterschiedlicher Art (numerisch und alphanumerisch) erfasst werden sollen, muß ein Stringarray verwendet werden.

Wenn in einer Variablen mehrere Informationen enthalten sein sollen, muß man die einzelnen Daten kennzeichnen, damit Sie später wieder aufgefunden werden können. Hier bietet sich ein Trennzeichen an, das in unserem Beispiel der Slash (/) sein soll. An Hand einer solchen Marke kann man die einzelnen Informationen sehr gut differenzieren.

Desweiteren muß die Reihenfolge der Dateneingabe festgelegt werden, von der im weiteren Verlauf nicht wieder abgewichen werden darf. In unserem Beispiel wird die Reihenfolge wie folgt festgelegt:

1. Eingabe der Bestellnummer
2. Eingabe der Artikelbezeichnung
3. Eingabe des Lagerortes
4. Eingabe des Sollbestandes
5. Eingabe des Ist-Bestandes

Diese Reihenfolge muß dem Anwender bei jeder Dateneingabe mitgeteilt werden. Doch zuerst muß eine Schleife angesteuert werden, die der Anzahl, der zu verarbeitenden Datensätze entspricht:

```
20 FOR I=1 TO 300
30 CALL CLEAR
40 PRINT 'REIHENFOLGE DER EINGABE:'
50 PRINT
60 PRINT 'BEST.NR/ART/LO/SOLL/IST'
70 PRINT
```

So, nun haben wir die Schleife angesteuert und dem Anwender mitgeteilt, in welcher Reihenfolge die Eingabe zu erfolgen hat.

Als nächstes folgt die eigentliche Dateneingabe:

```
80 INPUT 'DATA:':A$(I)
```

Nach den, oben beschriebenen Vereinbarungen, muß die Dateneingabe folgendermaßen aussehen:

```
DATA: 512-11-8064/Schrauben 8x4/A511/50/25
```

Nun kann man noch Eingabekontrollroutinen einbauen, die die Daten auf Gültigkeit überprüfen und eine nachträgliche Änderung erlauben. Auf eine solche Möglichkeit wird jetzt eingegangen.

Wir wollen jetzt aus der Speichervariablen die einzelnen Daten zur weiteren Verarbeitung trennen. In unserem Beispiel sollen die Daten zur Kontrolle und Änderung getrennt werden:

```
90 Q=POS(A$(I),'/',1)
95 Q$=SEG$(A$(I),1,Q-1)
```

Nachdem in Zeile 90 die Position des 1. Slashes festgestellt wurde, wird in Zeile 95 die erste Information der Stringvariablen Q$ zugewiesen. Jetzt muß überprüft werden, ob die

Information numerisch oder alphanumerisch ist. Da in unserem Beispiel zuerst eine alphanumerische Information erwartet wird, können wir im anderen Falle (Information ist rein numerisch) eine Fehlermeldung ausgeben und zur erneuten Eingabe zurückkehren:

```
100 FOR J=1 TO LEN(Q$)
110 X=ASC(SEG$(Q$,J,1))
120 IF X<48 THEN 150
125 IF X>57 THEN 150
130 NEXT J
140 GOTO 30
150 AN$=Q$
```

In Zeile 100 wird eine Schleife zur Abfrage der 1. Information angesteuert. Die Zeile 110 wandelt jedes Zeichen in den ASCII-Code um und in den Zeilen 120 und 125 wird getestet, ob das jeweilige Zeichen eine Ziffer ist. Ist ein Zeichen alphanumerisch (also keine Zahl) dann wird die Programmsteuerung nach 150 verzweigen, um die Information der Variablen AN$ zuzuweisen. Sollte die Information nach Durchlauf der Schleife rein numerisch sein, wird die Eingaberoutine neu aufgerufen (140). Wie oben erwähnt, kann man an dieser Stelle eine Fehlermeldung in das Programm mit einflechten.

Jetzt kann die zweite Information aus der Variablen A$(I) herausgefiltert werden:

```
160 Z=POS(A$(I),'/',Q+1)
170 Z$=SEG$(A$(I),Q+1,(Z-Q+1))
```

In gleicher Art und Weise wie vorher schon beschrieben wurde, wird in Zeile 160 die Position des zweiten Slashes festgestellt. Als Startpunkt der POS-Suchfunktion dient die Position des ersten Slashes aus Zeile 80, die in der Vari-

ablen Q gespeichert ist. Die Variable Z$ enthält nun die zweite Information. Jetzt erfolgt wieder die Überprüfung der Daten:

```
180 FOR J=1 TO LEN(Z$)
190 X=ASC(SEG$(Z$,J,1))
200 IF X<48 THEN 230
205 IF X>57 THEN 230
210 NEXT J
220 GOTO 30
230 ART$=Z$
```

In Zeile 180 wird wieder die Testschleife angesteuert. Da wiederum eine alphanumerische Information erwartet wird, geschieht die Prüfung in den Zeilen 200 und 205 in gleicher Weise wie vorher. Die Zeile 230 weist der Variablen ART$ die Daten aus Z$ zu.

Die dritte Information wird sondiert:

```
240 Q=POS(A$(I),'/',Z+1)
250 Q$=SEG$(A$(I),Z+1,(Q-Z+1))
```

Diese dritte Information wird als Inhalt den Lagerort enthalten. Auch dies ist eine alphanumerische Information und wird wie in den ersten beiden Fällen behandelt:

```
260 FOR J=1 TO LEN(Q$)
270 X=ASC(SEG$(Q$,J,1))
280 IF X<48 THEN 310
285 IF X>57 THEN 310
290 NEXT J
300 GOTO 30
310 LO$=Q$
```

In der vierten Information wird die Sollmenge enthalten sein. Hier wird sie ausgefiltert:

```
320 Z=POS(A$(I),'/',Q+1)
330 Z$=SEG$(A$(I),Z+1,(Z-Q+1))
```

Da die Sollmenge eine numerische Größe sein wird, muß nun die Überprüfung etwas anders aussehen:

```
340 FOR J=1 TO LEN(Z$)
350 X=ASC(SEG$(Z$,J,1))
360 IF X>48 THEN 361 ELSE 363
361 IF X<57 THEN 370
363 IF X=46 THEN 370
365 GOTO 30
370 NEXT J
380 SO=VAL(Z$)
```

Wenn in den Zeilen 360, 361 und 363 die Abfrage negativ ausfällt, wird die Programmsteuerung zur Zeile 365 weitergegeben, um von dort aus die Eingaberoutine neu einzuleiten.

Jetzt kann endlich die letzte Information herausgezogen werden. Da nach dieser letzten Information kein weiterer Slash mehr folgen wird, muß die Sondierung etwas anders als in den vorhergehenden Fällen aussehen:

```
390 Q$=SEG$(A$(I),Z+1,LEN(A$(I))-Z+1)
```

Wenn man von der Länge des Gesamtstrings die Position des letzten Slashes abzieht, erhält man die Länge der letzten gespeicherten Information.

Die Istmenge wird ebenfalls numerisch sein und kann wie oben getestet werden:

```
410 FOR J=1 TO LEN(Q$)
420 X=ASC(SEG$(Q$,J,1))
430 IF X>48 THEN 431 ELSE 433
431 IF X<57 THEN 440
433 IF X=46 THEN 440
435 GOTO 30
440 NEXT J
450 IST=VAL(Q$)
```

Nun können die Daten weiter verarbeitet werden, oder man kehrt direkt zur Eingabe der nächsten Daten über NEXT I zur Zeile 30 zurück. Vielleicht sollte man die Daten vor einer weiteren Eingabe nochmal visuell kontrollieren, indem sie auf dem Bildschirm ausgegeben werden:

```
460 CALL CLEAR
470 PRINT 'DATENSATZ';I
480 PRINT
490 PRINT
500 PRINT 'BEST.NR:';AN$
510 PRINT 'ARTIKEL:';ART$
520 PRINT 'LA-ORT :';LO$
530 PRINT 'SOLL   :';SO
540 PRINT 'IST    :';IST
550 PRINT
560 PRINT '****************************'
570 PRINT
580 PRINT ' DATEN O.K. (J/N) ?'
590 CALL KEY(0,K,S)
600 IF K=74 THEN 630
610 IF K=78 THEN 30
620 GOTO 590
630 NEXT I
```

Bitte beachten Sie, daß bei jeder Verarbeitung der gespeicherten Daten in oben beschriebener Art und Weise decodiert werden muß! Sicherlich ist der Programmieraufwand gegenüber der herkömmlichen Datenbehandlung erheblich, es wird jedoch bei größeren Datenmengen Speicherplatz gespart.

Da in einer einzigen Variablen immer ein ganzer Datensatz gespeichert ist, wird bei der Datenübertragung auf Kassettenrekorder eine Menge Zeit gespart, weil die Speicherschleife immer nur eine Variable enthält (siehe Kapitel 5.6).

Ein kleines Rechenbeispiel:

Wir wollen nun einmal die herkömmliche Datenverarbeitung mit der oben vorgestellten Methode rechnerisch vergleichen, um zu demonstrieren, wieviel Speicherplatz gespart werden kann:

Grundsätzliches am Anfang:

Den Berechnungen sollen folgende Daten zu Grunde liegen, die angeben, wieviel Speicherplatz vom System belegt wird.

Variablenname	belegt	2 Byte
Adresse der Variable	belegt	2 Byte

Bei diesem Rechenbeispiel richten wir uns nach den gleichen Eingabekriterien wie im obigen Beispielprogramm.

Herkömmliche Methode:

Wenn wir 300 Datensätze mit je 5 Informationen verwalten wollen, muß zur Datenspeicherung ein zweidimensionales

Stringarray angelegt werden:

```
DIM A$(300,5)
```

Dabei bedeuten:

A$(X,1)	=>	Bestellnummer
A$(X,2)	=>	Artikelbezeichnung
A$(X,3)	=>	Lagerort
A$(X,4)	=>	Sollmenge
A$(X,5)	=>	Istmenge

Daraus ergeben sich 300 x 5 = 1500 Speicherplätze, die im Rechner dimensioniert werden müssen. Versuchen Sie einmal, bei einem Datenverwaltungsprogramm eine Dimensionierung von 1500 Durchzuführen!

Diese 1500 Speicherplätze werden durch 1500 Variablenbezeichnungen benannt. Daraus kann man mit den bekannten Werten folgende Zahlen ermitteln:

Variablenbezeichnungen	x 2 Byte =>	1500 x 2 = 3000 Bytes	
Adressen	x 2 Byte =>	1500 x 2 = 3000 Bytes	
Benötigter Speicherplatz:			6000 Bytes

Sie sehen, daß durch den TI 99/4A alleine 6000 Bytes belegt werden, damit die Variablen und ihre Inhalte gefunden werden können. Wie sieht es nun bei unserer modifizierten Methode aus ?

Wenn wir uns nun die zweite, angeblich platzsparendere Methode betrachten, kommen wir auf die folgenden Werte:

Decodier- und Eingaberoutine	=>			800 Bytes
Variablenbezeichnungen	=>	300 x 2	=	600 Bytes
Adressen	=>	300 x 2	=	600 Bytes
Benötigter Speicherplatz:				2000 Bytes

Nun können Sie ganz deutlich sehen, daß unsere Routine nur ein Drittel des Speicherplatzes der herkömmlichen Methode benötigt. Dennoch muß hier gesagt werden, daß sich diese Routine erst bei 300 und mehr Datensätzen lohnt.

Hier ist noch einmal das komplette Listing der besprochenen Routine abgedruckt:

```
10 DIM A$(300)
20 FOR I=1 TO 300
30 CALL CLEAR
40 PRINT "REIHENFOLGE DER EINGABE:"
50 PRINT
60 PRINT "BEST.NR/ART/LO/SOLL/IST"
70 PRINT
80 INPUT "DATA:":A$(I)
90 Q=POS(A$(I),"/",1)
95 Q$=SEG$(A$(I),1,Q-1)
100 FOR J=1 TO LEN(Q$)
110 X=ASC(SEG$(Q$,J,1))
120 IF X<48 THEN 150
125 IF X>57 THEN 150
130 NEXTJ
140 GOTO 30
150 AN$=Q$
160 Z=POS(A$(I),"/",Q+1)
170 Z$=SEG$(A$(I),Q+1,(Z-Q+1))
180 FOR J=1 TO LEN(Z$)
190 X=ASC(SEG$(Z$,J,1))
200 IF X<48 THEN 230
205 IF X>57 THEN 230
210 NEXT J
220 GOTO 30
230 ART$=Z$
240 Q=POS(A$(I),"/",Z+1)
250 Q$=SEG$(A$(I),Z+1,(Q-Z+1))
260 FOR J=1 TO LEN(Q$)
270 X=ASC(SEG$(Q$,J,1))
280 IF X<48 THEN 310
285 IF X>57 THEN 310
290 NEXT J
300 GOTO 30
310 LO$=Q$
320 Z=POS(A$(I),"/",Q+1)
330 Z$=SEG$(A$(I),Q+1,(Z-Q+1))
340 FOR J=1 TO LEN(Z$)
350 X=ASC(SEG$(Z$,J,1))
360 IF X>48 THEN 361 ELSE 363
361 IF X<57 THEN 370
363 IF X=46 THEN 370
365 GOTO 30
370 NEXT J
380 SO=VAL(Z$)
390 Q$=SEG$(A$(I),Z+1,LEN(A$(I))-Z+1)
410 FOR J=1 TO LEN(Q$)
420 X=ASC(SEG$(Q$,J,1))
430 IF X>48 THEN 431 ELSE 433
431 IF X<57 THEN 440
433 IF X=46 THEN 440
435 GOTO 30
440 NEXTJ
450 IST=VAL(Q$)
460 CALL CLEAR
470 PRINT "DATENSATZ:";I
480 PRINT
490 PRINT
```

```
500 PRINT "BEST.NR:";AN$
510 PRINT "ARTIKEL:";ART$
520 PRINT "LA-ORT:";LO$
530 PRINT "SOLL:";SO
540 PRINT "IST:";IST
550 PRINT
560 PRINT "****************************"
570 PRINT
580 PRINT "DATEN O.K. (J/N) ?"
590 CALL KEY (O,K,S)
600 IF K=74 THEN 630
610 IF K=78 THEN 30
620 GOTO 590
630 NEXT I
```

5.7 Eingabeüberprüfung bei menugesteuerten Programmen

Es gibt verschiedene Methoden, um die Gültigkeit einer Eingabe bei Verwendung eines Menus zu überprüfen. In vielen Programmen, die diese Technik aufweisen, wird versäumt, eine Prüfung auf ein ungültiges Zeichen durchzuführen, so daß die Programmsteuerung undefiniert verzweigen kann. Im folgenden werden einige Möglichkeiten vorgestellt:

Zuerst wird ein Eingabemenu erzeugt, das den Beispielen als Grundlage dienen soll:

```
10 CALL CLEAR
20 PRINT 'BITTE WAEHLEN SIE:'
30 PRINT
40 PRINT
50 PRINT '1=> DATEI ERSTELLEN'
60 PRINT
70 PRINT '2=> DATEN VERARBEITEN'
80 PRINT
90 PRINT '3=> DATEN EINLESEN'
100 PRINT
110 PRINT '4=> DATEN ABSPEICHERN'
120 PRINT
130 PRINT '5=> PROGRAMMENDE'
140 PRINT
150 PRINT
160 CALL KEY(0,K,S)
170 IF S=0 THEN 160
```

Beispiel 1: In diesem Beispiel wird einfach getestet, ob das eingegebene Zeichen im zulässigen Bereich liegt:

```
180 IF K<49 THEN 160
190 IF K>53 THEN 160
200 ON K-48 GOTO X1,X2,X3,X4,X5.....
```

Da der Variablen K bei der Eingabe der ASCII-CODE des gerade gedrückten Zeichens zugewiesen wird, müssen die zulässigen Zeichen auch mit ihrem jeweiligen Code überprüft werden. Dies geschieht in den Zeilen 180 und 190. Fällt ein Zeichen nicht in den gültigen Bereich, wird das Programm nach 160 zurückkehren, um eine neue Eingabe zu erwarten. Ist die Überprüfung positiv, so kann zum entsprechenden Programmodul gesprungen werden. Dies muß nicht in der gezeigten Art und Weise geschehen. Die Zeichen könnten hier durchaus auch einzeln getestet werden, was jedoch einen größeren Speicherbedarf zur Folge hätte.

Beispiel 2: Eine, etwas elegantere Methode ist die Zuweisung der gültigen Zeichen in eine Stringvariable. Danach wird jedes Zeichen separat abgeprüft:

```
180 B$='12345'
190 K$=STR$(K-48)
200 FOR I=1 TO LEN(B$)
210 IF K$=SEG$(B$,I,1) THEN 240
220 NEXT I
230 GOTO 160
240 ON VAL(K$) GOTO X1,X2,X3,X4,X5.....
```

Sicherlich sind noch einige andere Methoden möglich, doch die beiden Beispiele werden dem Einsteiger ein guter Anhalt sein.

5.8 Runden von Zahlen

Bestimmt will jeder Anwender irgendwann einmal eine Zahlenkolonne in tabellarischer Form auf dem Bildschirm ausgeben lassen. Hier gibt es wegen des fehlenden BASIC-Befehles PRINT USING, bei der Darstellung Probleme. Wie man in TI-Basic diesen Befehl simulieren kann, wird später gezeigt. Wir wollen Ihnen hier zeigen, wie man ohne großen Aufwand alle Zahlen auf die jeweils gewünschten Nachkommastellen auf-bzw. abrunden kann. Dieser kleine Kniff kann zwar eine PRINT USING-Routine nicht ersetzen, ist aber sicherlich in vielen Fällen eine große Hilfe.

Beispiel : Folgende Zahl soll in eine Zahl mit zwei Nachkommastellen umgewandelt werden:

135.3287553

Um jetzt zwei Nachkommastellen zu erhalten, wird die Zahl mit 100 multipliziert, d.h. es werden die zwei ersten Nachkommastellen vor das Komma der Ursprungszahl geholt (will man die Zahl dreistellig haben, müsste man mit 1000 multiplizieren u.s.w.):

135.3287553 x 100 = 13532.87553

Da die gerundete Zahl keinen Rundungsfehler haben soll, muß man nun 0.5 hinzuaddieren:

13532.87553 + 0.5 = 13533.37553

Nun läst man einfach alle Nachkommastellen weg, indem man die Zahl in eine Integerzahl umwandelt und teilt die übrig-

gebliebene Zahl durch 100:

13533 : 100 = 135.33

Oben steht jetzt die gewünschte Zahl mit zwei Nachkommastellen, exakt gerundet. In BASIC müsste die Lösung folgendermaßen aussehen:

```
10 A=135.3287553
20 B=(INT(A*100+.5))/100
30 PRINT B
```

Sie sehen an diesem Beispiel, daß eine leidliche Formatierung auf diesem Wege recht schnell bewerkstelligt ist.

5.9 Wieviel Speicherplatz steht noch zur Verfügung ?

Bei der Eingabe eines Programmes möchte man oftmals den restlichen Speicherplatz, der noch verfügbar ist, wissen. Leider ist ein entsprechender Befehl in TI-BASIC nicht vorgesehen. Einfacher haben es die Besitzer von EXTENDED BASIC; dort ist eine Abfrage über die SIZE-Funktion möglich. Trotzdem gibt es einen kleinen Trick, der jedem TI-Anwender die Möglichkeit eröffnet, den noch verfügbaren Speicherplatz (RAM) zu erfahren:

```
1 RAM=RAM+7
2 GOSUB 1
>ENTER<
```

Nach wenigen Sekunden wird die kleine Routine mit einer BREAK IN - Meldung abgebrochen. Wenn Sie nun PRINT RAM eingeben, wird eine relativ genaue Angabe über den restlichen Speicherplatz erfolgen.

5.10 Programmschutz

Bekanntlich sind z. Zt. im EDV-Bereich noch keine ausreichenden juristische Schutzgesetze erarbeitet. Auf Grund dieser Tatsache kann und wird im Bereich der Programmentwicklung mit fremden Programmen oder Programmteilen Schindluder betrieben. Jeder, der ein Programm entwickelt hat, kann jedoch sein Programm mit einigen Kniffen so Kennzeichnen, daß er es jederzeit wiedererkennen kann. An Hand solcher Merkmale kann im Streitfall ein Programm als Eigentum nachgewiesen werden. Voraussetzung für eine solche Programmsicherung ist natürlich eine Dokumentation, damit nicht in Vergessenheit gerät, welches Programm, wie geschützt wurde.

Im folgenden werden einige dieser Tricks vorgestellt, die sehr einfach in ein Programm einzuflechten sind.

5.10.1 Das Passwort

Sehr oft wird zur Sicherung eines Programmes die Abfrage eines Passwortes eingebaut. In TI-BASIC kann der geübte Programmierer sehr schnell erkennen, um welches Wort es sich handelt, da hier ein LIST-Schutz nicht ohne weiteres möglich ist. Trotzdem soll an dieser Stelle eine Möglichkeit gezeigt werden. Bevor Sie den nach dem Listing folgenden Abschnitt lesen, sollten Sie vielleicht einmal versuchen, die Routine zu analysieren:

```
10 CALL CLEAR
20 PRINT 'PASSWORT:';
```

```
30 INPUT A$,A
40 T$='START'
50 T$=T$&CHR$(82)&CHR$(79)&CHR$(83)&CHR$(83)
60 T$=T$&'KASTANIE'
70 T$=T$&'MEHR'
80 IF A$<>SEG$(T$,A,4) THEN 110
90 CALL CLEAR
100 END
110 .
    .
    .
    .
```

Sicherlich ist es nicht ganz einfach, die obenstehende Routine zu durchschauen. Hier wird kein spezielles Passwort vordefiniert, sondern - aus oben genannten Gründen der Durchsichtigkeit - ein frei zusammengestellter Stringausdruck, aus dem später ein 4-stelliges Segment als Schlüsselwort dient. Natürlich muß der Anwender wissen, welcher Stringausdruck programmiert ist.

Derjenige, der das Programm benutzen will, muß bei der Abfrage nach dem Passwort in Zeile 30 zwei Werte eingeben:

1. Das eigentliche Passwort (4 Zeichen lang).
2. Die Startposition für SEG$.

Die Zeichenkette, die zur Definition des Schlüsselwortes dienen soll, wird in der Variablen T$ gespeichert. In den Zeilen 40 - 70 wird dieser String zugewiesen und zwar so, daß nicht auf den ersten Blick erkannt werden kann, um welchen Ausdruck es sich handelt. Folgende Stringfolge wird in unserem Beispiel in T$ stehen:

```
T$='STARTROSSKASTANIEMEHR'
```

Bei der Eingabe muß ein vierstelliges Wort eingegeben werden und die Anfangsposition des ersten Zeichens dieses Wortes.

Gültige Eingaben sind zum Beispiel:

```
STAR,1
ROSS,6
ASTA,11
EMEH,17
```

Jede beliebige Kombination aus T$ ist zur Anwendung als Passwort erlaubt. In Zeile 80 wird schließlich geprüft, ob ein rechtmäsiges Passwort eingegeben wurde.

Der Passwortschutz dient weniger zum Schutz des Programmes unberechtigter Weitergabe, als vielmehr zum Datenschutz. Sollten Sie z.B. mit einem Dateiverwaltungsprogramm Daten speichern, die Sie vor unberechtigtem Zugriff schützen wollen (z.B. vor Ihrer Frau), so empfiehlt sich die Sicherung des Programmes durch ein Passwort.

5.10.2 Fiktive Programmzeilen

Eine sehr wirkungsvolle Art, ein Programm als Ihr eigenes kenntlich zu machen, ist die Einflechtung von Programmsegmenten, die auf den Programmfluß keinen Einfluß nehmen. Dies kann auf unterschiedliche Art und Weise erfolgen:

A) Eine Möglichkeit ist das Einfügen von Wertzuweisungen, wobei die definierten Variablen im weiteren Programmfluß nicht wieder auftauchen dürfen.
   Beispiel:

```
10 X4=135
20 HF=INT(RND*100+.5)
```

Die Variablen X4 und HF sind Scheinvariablen und werden ohne jede Bedeutung definiert. Es ist sehr unwahrscheinlich, daß selbst ein geübter Programmierer diese kleinen Fallen entdeckt.

B) Die nächste Möglichkeit sind Verzweigungsbefehle, die immer wieder aus dem Programmlauf verzweigen (GOTO und GOSUB), um sofort - vielleicht in Verbindung mit der, unter A) vorgestellten Methode - zum Hauptprogramm zurückzukehren.
Beispiel:

```
10 GOTO 1050
20 GOSUB 10000
     .
     .
     .
     .
1050 X4=138
1060 GOTO 20
     .
     .
     .
     .
10000 RETURN
```

Es sei Ihrer Phantasie überlassen, wie diese wirkungsvollen Sicherungen ausgefeilt, bzw. weiterentwickelt werden. Viele Verwenden, nachdem Sie ein Programm entwickelt haben den RESEQUENCE-Befehl, um die Zeilennummern neu zu ordnen. Das Programm wird dadurch zwar übersichtlicher, ist aber um ein wertvolles Merkmal - die individuelle Zeilennummer - ärmer.

5.11 Der PRINT USING-Befehl

Die nachfolgend aufgeführte Routine simuliert den, in anderen Basic-Versionen vorhandenen BASIC-Befehl PRINT USING. Mit Hilfe dieser Routine ist man in der Lage, Zahlen in ein vernünftiges und einheitliches Format zu bringen. Besitzer des Moduls EXTENDED BASIC verfügen über diesen Befehl.

Zu Beginn gleich ein kleines Zahlenbeispiel:

Wenn Sie eine Zahlenkolonne darstellen wollen, sieht das in TI-BASIC ohne PRINT USING in der Regel so aus:

```
123.17
10.32
10245
1.9983
```

Bei dieser Darstellung ist es schwierig, die Zahlen zu überblicken. Mit der PRINT USING-Hilfsroutine kann man die Gesamtstellenlänge der größten möglichen Zahl und die Nachkommastellen bestimmen. Wenn wir nun die oben aufgeführten Zahlen mit zwei Nachkommastellen darstellen wollen, müssen wir als Gesamtstellenzahl 8 und für die Nachkommastellen 2 vorgeben. Die Hilfsroutine wird nun folgende Darstellung erzeugen:

```
  123.17
   10.32
10245.00
    1.99
```

Sie sehen, daß die Zahlenkolonne nun wesentlich übersicht-

licher dargestellt wird. Wenn Sie mit einem Drucker arbeiten wollen und Formulare ausdrucken, so ist eine solche Routine unbedingt erforderlich.

Im Folgenden eine kleine Bedienungsanleitung, damit Sie die Routine richtig anwenden können. Sie müssen, bevor die Routine angesprungen wird, einige Parameter übergeben, deren Bedeutung hier beschrieben wird:

FZ => Die zu formatierende Zahl (z.B. 1024.277363565)
NK => Nachkommastellen (z.B. 2 für das obige Beispiel)
VZ => Vorzeichen (bei '1' mit pos. Vorzeichen, bei '0' wird nur das negative Zeichen dargestellt)
FZ$=> Füllzeichen (Leerzeichen werden mit dem jeweils zugewiesenen Zeichen aufgefüllt. Z.B. '#')
ZA$=> Rückgabevariable (hier wird die Formatierte Zahl ins Hauptprogramm übergeben)

Die abgedruckte PRINT USING-Routine beginnt eigentlich erst ab Zeile 30000. Die Zeilen davor dienen zur Erprobung der Routine, in dem Sie einfach einige Zahlen eingeben und wieder ausgeben lassen.

```
1 REM UEBERGABEPARAMETER
2 REM FZ=>DIE ZU FORMATIERENDE ZAHL
3 REM NK=>NACHKOMMASTELLEN
4 REM VZ=>BEI '1' MIT POS. VORZEICHEN
5 REM GS=>GESAMTSTELLEN
6 REM FZ$=>FUELLZEICHEN
7 REM ZA$=>RUECKGABE DER FORMAT.ZAHL
8 PRINT
10 INPUT 'ZAHL:':FZ
```

```
20 INPUT 'NACHKOMMASTELLEN:':NK
30 INPUT 'GESAMTSTELLEN:':GS
40 INPUT '1= MIT + ZEICHEN:':VZ
50 INPUT 'FUELLZEICHEN:':FZ$
60 PRINT
70 GOSUB 30000
80 CALL CLEAR
90 PRINT ZA$
100 END
30000 IF GS<1 THEN 30500
30010 IF NK>GS-2 THEN 30500
30020 ZA$=STR$(ZA)
30030 FOR I=1 TO LEN(ZA$)
30035 IF SEG$(ZA$,I,1)='.' THEN 30070
30040 NEXT I
30050 ZA$=ZA$&'.'
30060 I=LEN(ZA$)
30070 ZA$=ZA$&'00000000000000000000'
30075 ZA$=SEG$(ZA$,1,I+NK)
30080 IF FZ>=0 THEN 30085 ELSE 30090
30085 ZA$=SEG$(ZA$,2,LEN(ZA$)-1)
30090 IF VZ=1THEN 30095 ELSE 30100
30095 ZA$='+'&ZA$
30100 IF LEN(ZA$)>GS THEN 30105 ELSE 30110
30105 X$='****************************'
30106 ZA$=SEG$(X$,1,GS)
30110 ZA$=SEG$(FZ$,1,GS-LEN(ZA$))&ZA$
30120 RETURN
30500 CALL CLEAR
30510 PRINT '******** FEHLER ********'
30520 STOP
```

Zum besseren Verständnis der obenstehenden Routine finden Sie im Folgenden ausführliche Erläuterungen zu den einzelnen Zeilen.

Programmbeschreibung:

Zeilennummer	Erläuterungen
10 - 60	In diesen Zeilen werden die Daten für die Übergabeparameter eingegeben.
70	Aufruf der Routine PRINT USING.
80 - 100	Ausgabe der formatierten Zahl.
30000	Wenn die Gesamtstellenzahl kleiner als 1 ist, wird die Routine mit einer Fehlermeldung abgebrochen (30500-30520).
30010	Wenn die Nachkommastellen grösser als die Gesamstellenzahl -2 sein soll, wird ebenfalls eine Fehlermeldung ausgegeben.
30020	Der Stringvariablen ZA$ wird die, zu formatierende Zahl aus ZA zugewiesen.
30030 - 30040	In dieser Schleife wird getestet, ob die eingegebene Zahl einen Dezimalpunkt hat. Wenn ja, verzweigt die Routine nach 30070.

Zeilennummer	Erläuterungen
30050	Wurde in der Schleife kein Dezimalpunkt entdeckt, wird dieser hier an die Variable ZA$ angehängt.
30060	Die Länge der Variablen ZA$ wird der Variablen I zugewiesen.
30070	An die Variable ZA$ werden hier 20 Nullen angehängt, die die unbelegten Nachkommastellen auffüllen sollen.
30075	Aus ZA$ wird jetzt ein Teil 'abgeschnitten', der der Länge der eingegebenen Zahl plus den Nachkommastellen entspricht.
30080	Wenn die Zahl im positiven Bereich. liegt, wird in Zeile 30085 die erste Stelle (Vorzeichenstelle) von ZA$ abgeschnitten.
30090	Ist als Vorzeichen eine '1' eingegeben worden, wird in Zeile 30095 ein '+' - Zeichen vor die Stringvariable ZA$ gebracht.
30100	Sollte die formatierte Zahl länger sein als am Anfang (30) definiert wurde, dann wird in Zeile 30106 ein Überlaufstring in Form

Zeilennummer	Erläuterungen
	von Sternchen der Variablen ZA$ zugewiesen.
30110	Bei Bedarf werden hier Füllzeichen zugewiesen.
30120	Ende der Routine und Rücksprung zum Hauptprogramm.
30500 - 30520	Ausgabe einer Fehlermeldung und Abbruch der Routine.

5.12 Sortieren von Zahlen

Irgendwann wird jeder Programmierer einmal mit dem Problem konfrontiert, Zahlen oder Nummern sortieren zu müssen. Wer eine Rangliste für seinen Sportverein mit dem Computer erstellen möchte, muß sich eine Sortierroutine überlegen. Je nach Sportart müssen die Werte aufsteigend (immer größer werdend) oder abfallend (immer kleiner werdend) sortiert werden.

Im Folgenden sollen beide Möglichkeiten einmal herausgearbeitet werden.

5.12.1 Sortieren nach aufsteigenden Werten

Zuerst müssen zwei Zählschleifen angesteuert werden, die der Anzahl der zu sortierenden Werte entsprechen:

```
100 WERTE=100
110 DIM A(100)
120 FOR S1=1 TO WERTE
130 FOR S2=1 TO WERTE
```

Danach wird in einer IF-Abfrage festgestellt, ob der Zahlenwert aus A(S1) größer ist, als der Wert aus A(S2). Kann die Abfrage mit 'JA' beantwortet werden, werden die Werte mit Hilfe von Zwischenspeichern ausgetauscht:

```
140 IF A(S1)>A(S2) THEN 150 ELSE 190
```

Ist nun der Wert aus A(S1) größer, wird dieser Wert im Zwischenspeicher X abgelegt:

```
150 X=A(S1)
```

Nun kann der Variablen A(S1) der kleinere Wert aus A(S2) zugewiesen werden:

```
160 A(S1)=A(S2)
```

Der, in der Variablen X stehende Wert, wird nun der Variablen A(S2) zugewiesen:

```
170 A(S2)=X
```

Nach diesem Tauschvorgang werden die beiden Zälschleifen abgeschlossen:

```
190 NEXT S2
200 NEXT S1
```

Wenn beide Schleifen voll durchlaufen sind, steht in der Variablen A(1) der kleinste und in der Varianlen A(100) der größte Wert.

5.12.2 Sortieren nach abfallenden Werten

Die Sortierroutine wird ähnlich aussehen wie beim Sortieren von aufsteigenden Werten. Der einzige Unterschied besteht in der IF-Abfrage in Zeile 140. Dort wird jetzt getestet, ob A(S1) kleiner als A(S2) ist, um im 'JA'-Fall wie unter 5.12.1 zu tauschen:

```
100 WERTE=100
110 DIM A(100)
120 FOR I=1 TO 100
130 A(I)=INT(RND*100)+1
140 NEXT I
150 FOR S1=1 TO WERTE
160 FOR S2=1 TO WERTE
170 IF A(S1)<A(S2) THEN 180 ELSE 220
180 X=A(S1)
190 A(S1)=A(S2)
200 A(S2)=X
220 NEXT S2
230 NEXT S1
240 FOR I=1 TO 100
250 PRINT A(I)
260 CALL KEY(0,K,S)
270 IF S=0 THEN 260
280 NEXT I
290 END
```

Nach Durchlauf der beiden Schleifen wird in A(1) der größte Wert stehen und in A(100) der kleinste.

Wir haben in unserem Beispielprogramm eine kleine Routine eingebaut (120 - 140), die der Variablen A(X) zufällig erzeugte Werte zwischen 1 und 100 zuweist. Wenn das Programm die Sortierroutine durchlaufen hat, werden die sortierten Werte auf dem Bildschirm einzeln ausgegeben. Durch Drücken irgendeiner Taste wird 'weitergeschaltet' und die nächstkleinere Zahl wird ausgegeben.

5.13 Alphabetisches Sortieren von Strings

Eine der bekanntesten und schnellsten Sortierroutinen für alphabetisches Sortieren in BASIC ist zweifellos die Routine Qicksort. Die meisten von Ihnen werden wahrscheinlich schon lange über diese Routine verfügen. Der Vollständigkeit halber finden Sie Quicksort hier noch einmal abgedruckt.

Die Routine sortiert Wörter bis zum letzten Buchstaben hindurch. Wenn Sie zum Beispiel folgende Begriffe in der gezeigten Reihenfolge eingeben,

BALLISTIK
TURM
AUERHAHN
KLAVIER
ROSE
BADEWANNE
SCHIFF

wird die Routine nach kurzer Zeit die Begriffe sortiert haben und auf dem Bildschirm wie folgt ausgeben:

AUERHAHN
BADEWANNE
BALLISTIK
KLAVIER
ROSE
SCHIFF
TURM

Sicherlich wird Ihnen dieses Hilfsprogramm eine wertvolle Hilfe bei den unterschiedlichsten Problemlösungen sein.

Um die Routine zu demonstrieren, wird in dem aufgelisteten Programm einer Variablen zufällig erzeugte Zeichenketten zugewiesen (200 Stück). Nach erstaunlich kurzer Wartezeit wird Ihnen die sortierte Liste auf dem Bildschirm ausgegeben. Den Autor dieser Routine können wir leider nicht nennen, da Nachforschungen unsererseits erfolglos geblieben sind. Zudem wurde im Laufe der Zeit durch verschiedene Anwender die Routine stark verändert.

```
1 REM ** SORTIERROUTINE QUICKSORT **
2 REM DIE EIGENTLICHE ROUTINE BEGINNT
3 REM AB ZEILE 20000.
4 REM DER SPEICHERVARIABLEN SO$(X)
5 REM WIRD IN DEN ZEILEN 10 - 90
6 REM ZUFAELLIG ERZEUGTE ZEICHENKETTEN
7 REM ZUGEWIESEN, UM DIE ROUTINE
8 REM ZU DEMONSTRIEREN.
10 DIM SO$(200),SO(200)
20 FOR I=1 TO 200
30 SO(I)=I
40 FOR J=1 TO 5
50 RANDOMIZE
60 TA$=CHR$(RND*24+65)
70 SO$(I)=SO$(I)&TA$
80 NEXT J
90 NEXT I
20000 DIM SRT(30,2)
20010 P=1
20020 SRT(1,1)=1
20030 SRT(1,2)=200
20040 S=SRT(P,1)
20050 N=SRT(P,2)
20060 P=P-1
20070 I=S
20080 J=N
20090 Q=SO(INT(S+N)/2)
```

```
20100 IF SO$(SO(I))>=SO$(Q) THEN 20130
20110 I=I+1
20120 GOTO 20100
20130 IF SO$(SO(J))<=SO$(Q) THEN 20170
20140 IF J<=S THEN 20170
20150 J=J-1
20160 GOTO 20130
20170 IF I>J THEN 20230
20180 U=SO(I)
20190 SO(I)=SO(J)
20200 SO(J)=U
20210 I=I+1
20220 J=J-1
20230 IF I<=J THEN 20100
20240 IF (N-I)<=(J-S) THEN 20310
20250 IF S>=J THEN 20290
20260 P=P+1
20270 SRT(P,1)=S
20280 SRT(P,2)=J
20290 S=I
20300 GOTO 20360
20310 IF I>=N THEN 20350
20320 P=P+1
20330 SRT(P,1)=I
20340 SRT(P,2)=N
20350 N=J
20360 IF N>S THEN 20070
20370 IF P<>0 THEN 20040
20380 FOR I=1 TO 200
20390 PRINT SO$(SO(I))
20400 NEXT I
20410 STOP
```

Was macht das Programm eigentlich ?

Das Prinzip der Sortierroutine ist recht schnell und einfach erklärt.
Während des Sortiervorganges werden den Strings, die in der Variablen SO$(X) stehen, entsprechend ihrer Wertigkeit im Alphabet, Ordnungsziffern vergeben. Die Zeichenkette, die im Alphabet am ersten steht, bekommt die Ziffer 1 und die letzte die Ziffer N (in unserem Beispiel 200). Es werden immer soviele Ordnungsziffern vergeben, wie Strings sortiert werden müssen.

Wenn die Sortierroutine abgeschlossen ist, kann man einfach durch Aufrufen der Ordnungsziffern, die in SO(X) stehen, die sortierte Liste ausgeben lassen.

KAPITEL 6: GRAPHIK AUF DEM TI 99/4A

Ihr TI verfügt über eine sehr leistungsfähige Graphik. In ersten Versuchen haben Sie sicherlich schon mit dieser Eigenschaft experimentiert. Für diejenigen unter Ihnen, die den TI vorwiegend zu spielerischen Zwecken nutzen wollen, ist eine intime Kenntnis der Graphikbefehle und -möglichkeiten unabdingbar. Hier bietet der TI 99/4A gegenüber vielen anderen, vergleichbaren Systemen, einen wesentlichen Vorteil: Während diese anderen Systeme über fest vorgegebene Graphikzeichen verfügen, die nicht verändert werden können, ist es beim TI möglich, jedes beliebige Zeichen frei zu definieren. Sie können sogar Ihren ganzen Zeichensatz umdefinieren. Doch bevor wir dazu kommen, sollen einmal die Befehle vorgestellt werden, die für graphische Zwecke benutzt werden können. Diese sind:

```
CALL SCREEN
CALL CHAR
CALL HCHAR
CALL VCHAR
CALL GCHAR
CALL COLOR
```

6.1 Die Routine CALL SCREEN

Mit dieser Routine können Sie die Farbe des Bildschirmes (SCREEN) verändern. Wenn Sie dieses integrierte Unterprogramm nutzen wollen, müssen Sie nach der SYNTAX noch einen Parameter übergaben, der dem jeweiligen Farbcode entspricht. Die Farbcodes können Sie im Anhang in der Farbcode-Tabelle

finden. Hier nun das vollständige Format:

```
CALL SCREEN (6)
```

Wenn Sie diese Zeile einmal im Direktmodus eingeben, können Sie sehen, daß der Bildschirm sofort die Farbe hellblau annimmt.

Wir wollen nun im Laufe dieses Kapitels - Stück für Stück - ein einfaches Graphikprogramm schreiben. Fangen wir einmal mit der Bildschirmfarbe an:

```
10 CALL CLEAR
20 INPUT 'SCREEN-FARBE:':FC
30 IF FC<1 THEN 10
40 IF FC>16 THEN 10
50 CALL SCREEN (FC)
```

In diesem Programmsegment geben Sie in Zeile 20 den Farbcode ein, den Sie gerne haben möchten. Die Zeilen 30 und 40 prüfen, ob der eingegebene Farbcode im zulässigen Bereich (1-16) liegt. Der CALL SCREEN-Befehl in Zeile 50 ändert schließlich den Bildschirm in die gewünschte Farbe.

6.2 Die Routine CALL COLOR

Mit diesem Befehl können Sie Vorder- und Hintergrundfarbe einer Zeichengruppe festlegen. Wie Sie sicherlich schon wissen, ist der Zeichensatz des TI 99/4A in Zeichengruppen aufgeteilt, die unter ihrer jeweiligen Nummer angesprochen werden können (siehe auch im Anhang). Hier sieht die vollständige SYNTAX so aus:

```
CALL COLOR (ZG,VF,HF)
```

Die Parameter in Klammern haben folgende Bedeutung:

ZG = Nummer der Zeichengruppe
VF = Vordergrundfarbcode
HF = Hintergrundfarbcode

Wir wollen in unserem Programm nun den Farbcode für eine bestimmte Zeichengruppe definieren:

```
60 PRINT
70 PRINT 'FARBE FUER ZEICHEN:'
80 PRINT
90 INPUT 'ZEICEHNGRUPPE:':ZG
100 IF ZG<1 THEN 90
110 IF ZG>16 THEN 90
120 INPUT 'VORDERGRUNDFARBE:':VF
130 IF VF<1 THEN 120
140 IF VF>16 THEN 120
150 INPUT 'HINTERGRUNDFARBE:':HF
160 IF HF<1 THEN 150
170 IF HF>16 THEN 150
180 CALL COLOR (ZG,VF,HF)
```

6.3 Die Routinen CALL HCHAR und CALL VCHAR

Mit diesen beiden Routinen können Sie beliebige Zeichen auf den Bildschirm bringen und je nach Befehl in horizontaler (CALL HCHAR) oder vertikaler (CALL VCHAR) Richtung wiederholen lassen. Die vollständige Befehlsfolge muß so eingegeben werden:

CALL HCHAR (S,Z,ZC,[WH])

oder

CALL VCHAR (S,Z,ZC,[WH])

Die Parameter bedeuten:

Z = Zeilennummer (1-24)

S = Spaltennummer (1-32)

ZC = Zeichencode des darzustellenden Zeichens.

WH = Wiederholungszahl. Diese Zahl muß nicht eingegeben werden und wiederholt ein Zeichen bei Bedarf in eine der beiden Richtungen. Fehlt dieser Wert, so wird nur ein Zeichen an der spezifizierten Bildschirmposition dargestellt.

Anmerkung: Da wir in jeder Spalte nur 28 Zeichen darstellen können, müssen Sie bei Eingabe der Spaltennummer beachten, daß die ersten und die letzten beiden Spaltennummern (1 - 2 und 31-32) im unsichtbaren Bereich liegen. Die Spaltennummer 3 bezieht sich z.B. auf die äußerst linke Bildschirmspalte.

In unserem Programm wollen wir nun den Bildschirm mit Sternchen einrahmen. Wenn Sie diese Zeichen in der, oben definierten Farbe darstellen wollen, müssen Sie als Zeichengruppe die '1' eingeben.

```
190 CALL VCHAR (1,3,42,24)
200 CALL VCHAR (1,30,42,24)
210 CALL HCHAR (1,3,42,28)
220 CALL HCHAR (24,3,42,28)
```

Unser Programm soll nun so erweitert werden, daß die Stern-

chen, die unseren Bildschirm umgeben, nach einigen Sekunden anfangen zu blinken. Zuerst erreichen wir mit einer Zählschleife, daß die Sternchen noch eine Weile auf dem Bildschirm zu sehen sind:

```
230 FOR I=1 TO 1000
240 NEXT I
```

Nun werden wir mit dem CALL COLOR-Befehl die Vorder- und Hintergrundfarbe der Sternchen so verändern, daß sie sich von der Bildschirmfarbe nicht mehr abheben werden, also für uns unsichtbar scheinen:

```
250 CALL COLOR (2,FC,FC)
```

Jetzt wird durch eine Zählschleife der Programmfluß verzögert, damit unsere Zeichen einen Moment im unsichtbaren Zustand verharren:

```
260 FOR I=1 TO 50
270 NEXT I
```

Als nächstes werden wir den Farbcode der Sternchen so verändern, daß die Hintergrundfarbe der Farbe des Bildschirmes entspricht und die Zeichen in der Farbe rot erscheinen:

```
280 CALL COLOR (2,9,FC)
```

Auch hier wird eine Zählschleife zur Verzögerung verwendet:

```
290 FOR I=1 TO 50
300 NEXT I
```

Wie oft unsere Sternchen blinken sollen, können wir in der IF-Abfrage (Zeile 320) bestimmen. Als Zähler benutzen wir

die Variable DL, die in der Zeile 310 nach jedem Blinken um 1 erhöht wird:

```
310 DL=DL+1
320 IF DL<=10 THEN 250
```

Wenn Sie das Programm, welches wir bis jetzt erstellt haben eintippen und mit RUN starten, wird der Rahmen aus Sternchen 10 mal blinken.

6.4 Zeichendefinition mit CALL CHAR

Mit der Systemrouine CHAR können Sie jedes beliebige Zeichen definieren. Für diesen Zweck sind die Zeichencodes 128 - 143 reserviert. Sie können jedoch auch jederzeit den Standart-Zeichensatz (32 - 127) umdefinieren. Die SYNTAX muß so aussehen:

```
CALL CHAR (ZC,'MUSTERCODE')
```

Die Parameter bedeuten hier:

ZC -- Ist der Zeichencode, unter dem wir später das frei definierte Zeichen mit HCHAR, VCHAR oder CHR$ aufrufen können.

Mustercode -- Hier wird in der Regel ein 16 Zeichen langer Stringausdruck eingegeben. Dieser Stringausdruck bestimmt die äußere Form des definierten Zeichens. Wie man diesen Stringausdruck zusammenstellt, um die 64 Bildpunkte eines Zeichens (8 x 8) ein - bzw. auszuschalten, erklären wir später.

Zur Entwicklung eines Zeichens ist es ratsam, ein 8 x 8 Kästchen großes Quadrat auf kariertes Papier zu zeichnen. Teilen Sie dieses Quadrat durch einen vertikalen Strich in zwei gleich große Blöcke zu je 4 Kästchen. Nun sind Sie bestens für den Entwurf eines neuen Zeichens gerüstet.

Wenn Sie Grundlagenkenntnisse über das hexadezimale und binäre Zahlensystem haben, werden Sie sich bei der Zusammenstellung des Stringausdruckes, der als Mustercode dient, sehr leicht tun, da der String aus hexadezimalen Zeichen besteht. Sollten Sie keine Kenntnisse über die angesprochenen Zahlensysteme haben, werden Sie dennoch - nach etwas Übung - ohne Schwierigkeiten Zeichen definieren können. Die folgende Tabelle wird Ihnen beim Entwerfen der Zeichen oder Figuren gute Dienste leisten.

Sehen wir uns einmal das Bitmuster eines Zeichens an:

```
L       :       R
* * * *:* * * *
* * * *:* * * *
* * * *:* * * *
* * * *:* * * *
* * * *:* * * *
* * * *:* * * *
* * * *:* * * *
* * * *:* * * *
```

Jedes Sternchen deutet an, das der betreffende Punkt eingeschaltet ist. Also haben wir es oben mit einem voll ausgefüllten Block zu tun. Beim Entwurf eines Zeichens betrachten wir immer zwei Blöcke, die in unserem Bild mit L und R gekennzeichnet sind. Jeder Block besteht immer aus vier Bildpunkten (oder Kästchen) und acht Zeilen. Im binären Zahlensystem können wir den Zustand einer Blockzeile sehr

einfach darstellen, indem für den eingeschalteten Punkt eine '1' und für den ausgeschalteten Punkt eine '0' gesetzt wird.
Beispiel:

Muster einer Blockzeile	. . * .
Binärzahl	0 0 1 0

Mit dem hexadezimalen Zahlensystem kann man nun vier Binärstellen durch ein einziges Zeichen ersetzen. Eine Tabelle der beiden Zahlensysteme soll Ihnen dies noch einmal verdeutlichen:

Binärcode	Hexadezimalzahl
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

Vielleicht ahnen Sie jetzt schon, warum der Stringausdruck

16 Zeichen umfassen muß!? Wie oben schon angedeutet, wird eine Punktzeile des neuen Zeichens durch zwei Ziffern oder Buchstaben aus dem hexadezimalen Zahlensystem beschrieben. Hierbei entspricht ein Zeichen immer den vier Punkten einer Blockzeile. Um einer Punktzeile das gewünschte Muster zu geben, wird immer mit der Blockzeile aus Block L begonnen.

Beispiel:

```
. . * *:* * . .   =>  0 0 1 1 : 1 1 0 0   =>  3 : C
-----------------------------------------------------
  L   :   R             L   :   R           L : R
```

In unserem Beispiel müßte der Musterstring mit '3C' beginnen. Zur Veranschaulichung wollen wir nun den Buchstaben A mit dem Zeichencode 128 neu definieren:

```
. . . *:* . . .   =>  0 0 0 1:1 0 0 0   =>  1 : 8
. . * *:* * . .   =>  0 0 1 1:1 1 0 0   =>  3 : C
. * * .:. * * .   =>  0 1 1 0:0 1 1 0   =>  6 : 6
* * . .:. . * *   =>  1 1 0 0:0 0 1 1   =>  C : 3
* * * *:* * * *   =>  1 1 1 1:1 1 1 1   =>  F : F
* * * *:* * * *   =>  1 1 1 1:1 1 1 1   =>  F : F
* * . .:. . * *   =>  1 1 0 0:0 0 1 1   =>  C : 3
* * . .:. . * *   =>  1 1 0 0:0 0 1 1   =>  C : 3
  L     R             L     R             L   R
```

Wenn Sie jetzt das Punktmuster mit dem umgesetzten Binärcode vergleichen, werden Sie die enge Verbundenheit zwischen Entwurf und Zahlencode entdecken. Unser Musterstring wird so aufgestellt: '183C66C3FFFFC3C3'

Der Buchstabe 'A' kann jetzt definiert werden:

```
CALL CHAR(128,'183C66C3FFFFC3C3')
```

Um Ihnen das Entwerfen von Zeichen etwas einfacher zu machen, stellen wir Ihnen im Folgenden ein Programm vor, mit dessen Hilfe Sie Ihre Sonderzeichen auf dem Bildschirm sehr leicht definieren können. Sie können diese Hilfsroutine in jedes beliebige Programm mit einflechten, um dort auf einfachem Wege Ihre Zeichen zu entwerfen. Zur Kontrolle gibt die Routine - nach der Definition - das Zeichen auf dem Bildschirm aus. Wenn alle Ihre Zeichen definiert sind, können Sie die Routine ruhig mit NEW löschen, ohne das ein Zeichen verloren geht.

```
10 DIM A$(8),B$(16),C$(16)
20 CALL CLEAR
30 PRINT 'SIE KOENNEN NUN JEDES'
40 PRINT 'ZEICHEN LEICHT DEFINIEREN,'
50 PRINT 'INDEM SIE AUF DEM BILD-'
60 PRINT 'SCHIRM EINFACH DAS MUSTER'
70 PRINT 'DES GEWUENSCHTEN ZEICHENS'
80 PRINT 'EINGEBEN. Z.B.:'
90 PRINT
100 PRINT '      ..*..***'
110 PRINT ::::
120 PRINT 'DRUECKE EINE TASTE!'
130 CALL KEY(0,K,S)
140 IF S=0 THEN 130
150 CALL CLEAR
160 FOR I=1 TO 8
170 INPUT 'MUSTER:':A$(I)
180 IF LEN(A$(I))<1 THEN 170
190 IF LEN(A$(I))>8 THEN 170
200 NEXT I
210 RESTORE
220 FOR I=1 TO 16
230 READ B$(I),C$(I)
240 NEXT I
```

```
250 FOR I=1 TO 8
260 FOR J=1 TO 16
270 V1$=SEG$(A$(I),1,4)
280 V2$=SEG$(A$(I),5,4)
290 IF V1$=B$(J) THEN 300 ELSE 310
300 T$=C$(J)
310 IF V2$=B$(J) THEN 320 ELSE 330
320 T1$=C$(J)
330 NEXT J
340 M$=M$&T$&T1$
350 T$=' '
360 T1$=' '
370 NEXT I
380 CALL CLEAR
390 INPUT 'WELCHER ZEICHENCODE:':ZC
400 CALL CLEAR
410 CALL CHAR (ZC,M$)
420 PRINT CHR$(ZC)
430 M$=' '
440 PRINT ::::'NOCH EIN ZEICHEN?'
450 CALL KEY(0,K,S)
460 IF S=0 THEN 450
470 IF CHR$(K)='J' THEN 150
480 CALL CLEAR
490 END
500 DATA ....,0,...*,1,..*.,2,..**,3,.*..,4
510 DATA .*.*,5,.**.,6,.***,7,*...,8,*..*,9
520 DATA *.*.,A,*.**,B,**..,C,**.*,D,***.,E
530 DATA ****,F
```

Um nun unser kleines Graphikprogramm, das wir unter 6.1 begonnen haben, zu erweitern, sollen die Sternchen, die unseren Bildschirm einrahmen, durch ausgefüllte Blöcke ersetzt werden, so daß wir einen vollen Rahmen erhalten. Hierzu muß dem Zeichencode des Sternchens (42) ein anderes Muster zugewiesen werden. An einem ausführlichen Beispiel soll noch einmal gezeigt werden, wie der zur Routine CALL CHAR gehörende Stringausdruck erstellt wird:

```
* * * *:* * * *    =>    1 1 1 1:1 1 1 1    =>    F : F
* * * *:* * * *    =>    1 1 1 1:1 1 1 1    =>    F : F
* * * *:* * * *    =>    1 1 1 1:1 1 1 1    =>    F : F
* * * *:* * * *    =>    1 1 1 1:1 1 1 1    =>    F : F
* * * *:* * * *    =>    1 1 1 1:1 1 1 1    =>    F : F
* * * *:* * * *    =>    1 1 1 1:1 1 1 1    =>    F : F
* * * *:* * * *    =>    1 1 1 1:1 1 1 1    =>    F : F
* * * *:* * * *    =>    1 1 1 1:1 1 1 1    =>    F : F
```

Der Musterstring wird so aussehen: 'FFFFFFFFFFFFFFFF'

Nun die ergänzenden Zeilen zu unserem Programm:

```
330 CALL CHAR(42,'FFFFFFFFFFFFFFFF')
340 REM FARBSPIEL
350 FOR I=1 TO 16
360 F=INT(RND*I)+1
370 CALL COLOR(2,F,F)
380 NEXT I
390 CALL COLOR(2,FC,FC)
```

Unser Programm wird in Zeile 330 das Sternchen in einen ausgefüllten Block umdefinieren. In den Zeilen 350 - 380 wird unser, jetzt ausgefüllter Rahmen, 16 mal die Farbe wechseln, und in Zeile 390 werden alle Zeichen der Zeichengruppe 2 - wozu auch unser ausgefüllter Block zählt - in die Ur-

6.5 Die Routine CALL GCHAR

Mittels CALL GCHAR (engl.: get character) ist es Ihnen möglich, eine beliebige Bildschirmposition auf ein Zeichen zu testen, dessen ASCII-Code der Variablen im zugehörigen Klammerausdruck zugewiesen wird. Die korrekte SYNTAX dieses Befehles ist:

CALL GCHAR (Z,S,AC)

Die Parameter in der Klammer haben folgende Bedeutung:

Z - Zeilennummer (1 - 24)
S - Spaltennummer (1 - 32)
AC - Rückgabe des ASCII-Codes von dem Zeichen, welches sich an der spezifizierten Bildschirmposition befindet.

Wenn Sie z.B. einen vollkommen leeren Bildschirm haben, wird der Variablen AC immer der ASCII-Code des Leerzeichens zugewiesen (= 32).

Anmerkung: Auch hier müssen Sie darauf achten, daß bei der Spaltenangabe die ersten und letzten beiden Ziffern im unsichtbaren Bereich liegen.

In unserem Graphikprogramm wollen wir nun überall dort, wo sich innerhalb des Rahmens ein Leerzeichen befindet, einen ausgefüllten Block plazieren. Um das ganze etwas plastischer zu gestalten, wird die Spalten - und Zeilennummer jeweils durch eine Zufallszahl ermittelt. Es sollen insgesamt 100 Blöcke im Rahmen dargestellt werden. Sind alle 100 Blöcke ausgegeben, wird die CALL CHAR-Routine auf dem Bildschirm nach diesen Blöcken suchen und sie wieder durch Leerzeichen

ersetzen:

```
400 FOR I=1 TO 100
410 S=INT(RND*(27-5+1))+5
420 Z=INT(RND*(22-3+1))+3
430 CALL GCHAR(Z,S,AC)
440 IF AC<>32 THEN 410
450 CALL HCHAR(Z,S,42)
460 NEXT I
470 FOR I=1 TO 100
480 S=INT(RND*(27-5+1))+5
490 Z=INT(RND*(22-3+1))+3
500 CALL GCHAR(Z,S,AC)
510 IF AC=32 THEN 480
520 CALL HCHAR(Z,S,32)
530 NEXT I
540 END
```

In den Zeilen 400 - 460 wird dort, wo sich ein Leerzeichen befindet, ein Block ausgegeben. Die Schleife (400) legt fest, wieviele Blöcke ausgegeben werden sollen. Die Zeilen 470 - 530 löschen allmählich den Bildschirm, indem durch CALL GCHAR (500) ein Block gesucht und gelöscht wird. Die Schleife (470) legt hier fest, wieviele Blöcke gelöscht werden sollen.

Unser kleines Graphikprogramm hat hier sein Ende gefunden. Wir hoffen, daß Ihnen dadurch einige geistige Anstöße für Ihre weitere Programmierung gegeben werden konnten. In unseren weiteren Beispielen wollen wir Ihnen nun zeigen, wie man einzelne Zeichen auf dem Bildschirm bewegen kann.

6.6 Bewegte Graphik mit dem TI 99/4A

Wir wollen nun versuchen, ein Zeichen auf dem Bildschirm zu bewegen. Bewegte Graphiken benötigen hauptsächlich diejenigen unter Ihnen, die sich vornehmlich mit der Programmierung von Spielen beschäftigen.

Bedenken Sie, daß eine bewegte Graphik immer auf einer optischen Täuschung des menschlichen Auges beruht, da ein Zeichen an einer beliebigen Position dargestellt, einen Moment in dieser Position verharrt, wieder gelöscht und an der nächsten Position dargestellt wird. Um dies zu demonstrieren, wollwn wir einmal den Buchstaben A horizontal über den Bildschirm wandern lassen:

```
10 CALL CLEAR
20 S=3
30 CALL HCHAR(10,S,65)
40 REM ** VERZOEGERUNG **
50 FOR I=1 TO 50
60 NEXT I
70 REM ** ZEICHEN LOESCHEN **
80 CALL HCHAR(10,S,32)
90 REM ** AN NEUER POSITION DARSTELLEN **
100 S=S+1
110 IF S<=30 THEN 30
120 END
```

Der Buchstabe A wird nun in der 10. Zeile ganz langsam und sprunghaft von links nach rechts wandern. Verändern Sie ruhig einmal die Werte der Verzögerungsschleife (50 - 60), um die Bewegungsgeschwindigkeit zu variieren. Am schnellsten wird der ganze Prozess, wenn Sie die Zeilen 50 und 60 ganz

weglassen. Um der spielerischen Anwendung der bewegten Graphik noch näher zu kommen, wollen wir die Bewegung des Buchstabens A einmal mit der Tastatur beeinflussen.

Zur Steuerung in die verschiedenen Richtungen verwenden Sie bitte folgende Tasten:

S => nach links
D => nach rechts
E => nach oben
X => nach unten

```
10 CALL CLEAR
20 Z=12
30 S=16
40 CALL HCHAR(Z,S,65)
50 CALL KEY(0,K,SP)
60 K$=CHR$(K)
70 IF K$='E' THEN 80 ELSE 90
80 Z=Z-1
85 GOTO 160
90 IF K$='X' THEN 100 ELSE 110
100 Z=Z+1
105 GOTO 160
110 IF K$='S' THN 120 ELSE 130
120 S=S-1
125 GOTO 160
130 IF K$='D' THEN 140 ELSE 150
140 S=S+1
145 GOTO 160
150 GOTO 50
160 CALL CLEAR
170 CALL HCHAR(Z,S,65)
180 GOTO 50
```

Das vorstehende Programm wird solange laufen, bis Sie es mit < FCTN 4 > unterbrechen. Um das Programm nicht übermäßig kompliziert zu machen, wurde keine Überprüfung der Daten mit eingebaut, was zur Folge hat, daß das Programm nach Überschreiten der zulässigen Bereiche (Zeile 24, Spalte 32), abstürzt.

Die Zeilen 10 - 40 positionieren den Buchstaben A in der Mitte des Bildschirmes. Danach wird getestet, welche der Steuertasten gedrückt wurde (50 - 150) und A wird an der neu berechneten Position dargestellt (160 - 180).

Wenn Sie diese Zeilen lesen, sind Sie gleichzeitig am Ende des Kapitels 'Graphik' angekommen. Wir hoffen, daß Sie wertvolle Neuerkenntnisse hinzugewonnen haben und Ihre Programme mit einer guten Graphik - im Rahmen des TI-BASIC - versehen können.

KAPITEL 7: DER GUTE TON IM TI

Zu einem richtigen Tele-Spiel gehört ein deftiger Ton. Wie Sie Ihre Graphikprogramme 'vertonen' können, wollen wir Ihnen in diesem Kapitel zeigen.

Der TI 99/4A verfügt nicht nur über eine hervorragende Graphik, er kann auch sehr gut Musik oder Geräusche fabrizieren. Eine Frequenzbreite von vier Oktaven deutet darauf hin, daß Ihr Computer ausgesprochen musikalisch sein kann, wenn man es versteht, ihm zur rechten Zeit die richtigen Töne zu entlocken. Damit ihm überhaupt Töne entlockt werden können, müssen Sie zuerst einmal den Befehl dazu kennen:

```
CALL SOUND (TD,F1,LS1,F2,LS2,F3,LS3,F4,LS4)
```

Mit dieser Befehlsfolge wird die Routine SOUND aufgerufen. Im zugehörigen Klammerausdruck können Sie bis zu 4 verschiedene Töne (Frequenzen: F1 - F4) und Lautstärken (LS1 - LS4) mit einer bestimmten Tondauer (TD) festlegen. Die entsprechenden Werte für die Frequenzen entnehmen Sie bitte der Tontabelle im Anhang. Im Folgenden sollen die Besonderheiten der einzelnen Werte aufgezeigt werden:

Tondauer -- Als Tondauer wird eine ganze Zahl im Bereich von 1 bis 4250 oder von -1 bis -4250 eingegeben. Die Zahl 1 entspricht einer Tondauer von ca. 1/1000 stel Sekunde. Das bedeutet, daß der längste Ton ca. 4.25 Sekunden (4250) dauert. Während der TI Musik macht, läuft das Programm ohne anzuhalten weiter. Stößt der Programmzeiger auf eine neue CALL SOUND-Anweisung, so wird zuerst die vorhergehende

Anweisung durchgeführt. Diese Eigenschaft können Sie durch die Eigaben eines negativen Vorzeichens umgehen. Dann wird bei Erreichen einer neuen SOUND - Anweisung die alte gestoppt, um sofort die neue auszuführen.

Frequenz -- Alle positiven Werte erzeugen einen Ton, während negative Werte ein Geräusch verursachen. In einem einzigen SOUND - Befehl können Sie bis zu drei Töne und ein Geräusch gleichzeitig erzeugen. In der folgenden Tabelle finden Sie die einzelne Bedeutung der negativen Werte für Geräusche:

WERT	BEDEUTUNG
-1	Pulsierendes Rauschen (TYP 1)
-2	Pulsierendes Rauschen (TYP 2)
-3	Pulsierendes Rauschen (TYP 3)
-4	Pulsierendes Rauschen, wobei die dritte angegebene Tonfrequenz verändert wird.
-5	Konstantes (weißes) Rauschen 1
-6	Konstantes (weißes) Rauschen 2
-7	Konstantes (weißes) Rauschen 3
-8	Konstantes (weißes) Rauschen mit Änderung der dritten angegebenen Tonfrequenz.

Lautstärke -- Hier können Sie alle ganzzahligen Werte von 0 bis 30 eingeben, wobei '0' die größte Lautstärke ist.

Ein kleines Programm, mit dem Sie mit Tönen experimentieren

können, soll Ihnen den Einstieg - falls Sie überhaupt noch Einsteiger sind - in die Tonprogrammierung etwas erleichtern:

Das Programm fragt Sie zunächst, wieviele Töne in einer SOUND-Anweisung ertönen sollen (max. 3). Danach geben Sie Tondauer, Frequenz(en) und Lautstärke(n) ein:

```
10 CALL CLEAR
20 INPUT 'WIEVIELE TOENE:':ZT
30 IF ZT<1 THEN 10
40 IF ZT>3 THEN 10
50 CALL CLEAR
60 INPUT 'TONDAUER       :':TD
70 PRINT
80 FOR I=1 TO ZT
90 PRINT 'FREQUENZ';I;TAB(16);':';
100 INPUT F
110 PRINT 'LAUTSTAERKE';I;TAB(16);':';
120 INPUT LS
130 PRINT
140 FR(I)=F
150 L(I)=LS
160 NEXT I
170 CALL CLEAR
180 ON ZT GOTO 190,210,230
190 CALL SOUND(TD,FR(1),L(1))
200 GOTO 250
210 CALL SOUND(TD,FR(1),L(1),FR(2),L(2))
220 GOTO 250
230 CALL SOUND(TD,FR(1),L(1),FR(2),L(2),
    FR(3),L(3))
250 PRINT 'NOCHMAL (J/N) ?'
260 CALL KEY(0,K,S)
270 IF S=0 THEN 260
```

```
280 K$=CHR$(K)
290 IF K$='J' THEN 10
300 END
```

In Telespielen wird am häufigsten ein Geräusch (Explosion) zur Untermalung verwendet. Unser folgendes Beispielprogramm zeigt Ihnen die verschiedenen Möglichkeiten zur Erzeugung von Geräuschen. Gleichzeitig werden auf dem Bildshirm die jeweils zugehörigen Werte angezeigt, damit die Geräusche, die Ihnen zusagen, schriftlich fixiert werden können:

```
10 CALL CLEAR
20 PRINT 'BITTE KENNZIFFER EINGEBEN:'
30 PRINT ::::
40 PRINT '[1] - PERIOD. RAUSCHEN 1'
50 PRINT '[2] - PERIOD. RAUSCHEN 2'
60 PRINT '[3] - PERIOD. RAUSCHEN 3'
70 PRINT '[4] - WEISSES RAUSCHEN 1'
80 PRINT '[5] - WEISSES RAUSCHEN 2'
90 PRINT '[6] - WEISSES RAUSCHEN 3'
100 PRINT ::::'BITTE WAEHLEN SIE!'
110 CALL KEY(0,K,S)
120 IF S=0 THEN 110
130 IF K<49 THEN 110
140 IF K>54 THEN 110
150 IF K-48>3 THEN 180
160 RA=-(K-48)
170 GOTO 200
180 RA=-(K-48+1)
200 CALL CLEAR
210 INPUT 'TONDAUER      :':TD
220 INPUT 'LAUTSTAERKE  :':LS
230 CALL CLEAR
240 CALL SOUND(TD,RA,LS)
```

```
250 PRINT 'TONDAUER      :';TD
260 PRINT 'FREQUENZ      :';RA
270 PRINT 'LAUTSTAERKE   :';LS
280 PRINT ::::'NOCHMAL (J/N) ?'
290 CALL KEY(0,K,S)
300 IF S=0 THEN 290
310 K$=CHR$(K)
320 IF K$='J' THEN 10
330 END
```

Mit diesem Programm können Sie sehr leicht alle Geräusche, die Sie für ein Programm verwenden wollen, austüfteln.

Zum Abschluß dieses Kapitels noch ein kleines Programm, das Ihnen sämtliche Töne, die mit dem TI 99/4A möglich sind, vorspielt:

```
10 CALL CLEAR
20 DIM F(48)
30 FOR I=1 TO 48
40 READ F(I)
50 NEXT I
60 FOR I=1 TO 48
70 CALL SOUND(500,F(I),0)
80 NEXT I
90 FOR I=48 TO 1 STEP -1
100 CALL SOUND(500,F(I),0)
110 NEXT I
120 END
130 DATA 110,116,123,130,138,164,155,164
140 DATA 174,184,195,207,220,233,246,261
150 DATA 277,293,311,329,349,369,391,415
160 DATA 440,466,493,523,554,587,622,659
170 DATA 698,739,783,830,880,932,987,1046
180 DATA 1108,1174,1244,1318,1396,1479,1567
190 DATA 1661
```

KAPITEL 8: MöGLICHKEITEN DES EXTENDED BASIC

Mit dem Modul EXTENDED BASIC können Sie Ihre Programmiermöglichkeiten erheblich erweitern. Viele Befehle, die Sie bisher nur aus unserer Beschreibung über Microsoftbasic her kennen, haben Sie jetzt zur Verfügung. In diesem Kapitel wollen wir Ihnen kurz erläutern, welche zusätzlichen Befehle bereitgestellt werden. Dieses Kapitel ist auch für diejenigen unter den Lesern interessant, die über dieses Modul nicht verfügen, da hier Hilfen gegeben werden, die das Umschreiben der Programme aus diesem Buch, die in EXTENDED BASIC geschrieben sind, wesentlich erleichtern.

Selbstverständlich haben Sie alle Befehle, die in TI-BASIC vorrätig sind, auch weiterhin zur Verfügung. Zusätzlich bietet das EXTENDED BASIC Modul folgende Befehle:

ACCEPT AT	CHARPAT(U)	CHARSET(U)	COINC(U)
DELSPRITE(U)	DISPLAY AT	DISTANCE(U)	ERR
IMAGE	INIT(U)	LINK(U)	LINPUT
LOAD(U)	LOCATE(U)	MAGNIFY(U)	MAX
MERGE	MIN	MOTION(U)	ON ERROR
ON WARNING	PATTERN(U)	PEEK(U)	PI
POSITION(U)	PRINT USING	REC	RPT$
SAY(U)	SIZE	SPGET(U)	SPRITE(U)
SUB	SUBEND	SUBEXIT	VERSION(U)

Alle mit (U) gekennzeichneten Befehle sind Unterprogrammroutinen und mit dem Zusatz >CALL< aufzurufen.

Zu den oben vorgestellten Befehlen sind viele Statements des TI-BASIC wesentlich erweitert bzw. verbessert worden.

8.1 Besonderheiten des EXTENDED BASIC

Wenn Sie dieses Modul anwenden wollen, müßen Sie folgende Besonderheiten beachten, die aus einem einzigen Nachteil aber viele Vorteilen bestehen:

A) Der Nachteil:

Wenn Sie das Modul in die Konsole eingesteckt haben, hat sich der Programmspeicher um ca. 900 BYTE verkleinert. Dies läßt sich jedoch durch die rationellen Programmiertechniken (Mehrfachzeilenbelegung) voll ausgleichen.

B) Mehrfachzeilenbelegung:

Sie können in einer Zeile mehrere Befehle eingeben, wobei die einzelnen Statements durch zwei Doppelpunkte (::) getrennt werden müssen.

Beispiel: 100 A=50 :: B=100 :: C=-90

Eine Mehrfachzeilenbelegung ist immer dann empfehlenswert, wenn dadurch die Programmstruktur, durch die ein Programm leichter zu überschauen ist, verbessert werden kann. Als Anwendungsbeispiele könnte man hier FOR....NEXT-Schleifen oder IF....THEN-Abfragen nennen:

```
100 FOR I=1 TO 20 :: READ A$ :: NEXT I
200 IF A=B THEN PRINT 'XXX' :: A=1 :: GOSUB 80 :: GOTO 10
```

C) Mehrfachzuweisung von gleichen Werten:

Sollen mehrere gleiche Werte zugewiesen werden, so können Sie dies in einer Zeile tun, indem Sie die Variablen einfach durch Komma trennen. Der Vorteil gegenüber TI-BASIC ist im folgenden Beispiel klar ersichtlich:

EXTENDED BASIC	TI-BASIC
100 A,B,C,D,E,F=100	100 A=100
	110 B=100
	120 C=100
	130 D=100
	140 E=100
	150 F=100

Beide Programmierweisen erfülen den selben Zweck. In TI-BASIC sind jedoch wesentlich mehr Programmzeilen (gleichzeitig mehr Speicherplatz) erforderlich.

D) Programmgeschwindigkeit:

Bei der Verwendung des EXTENDED BASIC Moduls werden, laut TI, alle Programme um den Faktor 2 schneller. Wie dies realisiert wird, konnten wir leider nicht in Erfahrung bringen.

E) Programmabtastung:

Wenn Sie in TI-BASIC ein Programm mit RUN starten, dauert es immer eine ganze Weile, bis das Programm endlich losläuft. Diese zeitliche Verzögerung liegt an der Programmabtastung: Nach der Eingabe von RUN tastet der Programmzeiger alle Programmzeilen auf Fehler und zur Reservierung von Speicherplatz ab; erst wenn alles in Ordnung ist, kann das Programm gestartet werden. Durch zwei Befehle ist es nun möglich, dem Programmzeiger mitzuteilen, daß bestimmte Programmteile n i c h t abgetastet werden sollen:

```
!@P+  =>  Einschalten der Abtastsperre
!@P-  =>  Ausschalten der Abtastsperre
```

Es ist nicht empfehlenswert, das gesamte Programm von einer Abtastung auszusparen, da sonst in Programmteilen, in denen Werte zugewiesen oder Variablen definiert werden, keine Speicherplatzreservierung erfolgt. Dies kann dazu führen, daß Ihr Rechner mitten im Programmlauf mit der Fehlermeldung 'OUT OF MEMORY' abstürtzt.

Beispiel:

```
10 !@P+
20 PRINT 'DIESE ZEILEN SIND VON'
30 PRINT 'DER ABTASTUNG NICHT'
40 PRINT 'BETROFFEN!!!'
50 !@P-
60 DIM A(100)
.
.
.
```

In unserem Beispiel erfolgt eine Abtastung erst ab Zeile 60ff.

F) Kommentarzeilen:

Das übliche Statement für Kommentarzeilen ist REM. Bei EXTENDED BASIC können Sie zsätzlich das Ausrufezeichen (!) als Kennzeichnung eines Schlußkommentars verwenden.

Diese Möglichkeit verwendet man sinnvollerweise dort, wo man ein Programm optimal strukturieren möchte. Durch zusätzliche Kommentare in Programmzeilen wird es für fremde Anwender leichter, das betreffende Programm nachzuvollziehen.

Beispiel:

```
10 FOR I=1 TO 500 ! START ZAEHLSCHLEIFE
20 X=X+1
30 NEXT I ! ENDE DER ZAEHLSCHLEIFE
40 H=RND*10 ! ZUFALLSZAHL
```

G) Frei definierbare Zeichencodes:

Wie Sie wissen, sind bei TI-BASIC die Zeichencodes 128 - 159 für frei definierbare Zeichen reserviert. Dieser Bereich wird bei EXTENDED BASIC etwas eingeschränkt, da ein Teil dieses Speicherbereiches durch SPRITES (frei bewegliche, unabhängige Figuren) belegt wird. Für unsere Zeichen in EXTENDED BASIC sind nunmehr die Codes 128 - 143 verfügbar.

Die wesentlichen Besonderheiten des EXTENDED BASIC haben wir nun angesprochen und können uns im Folgenden den Befehlen widmen.

8.2 Befehle des EXTENDED BASIC

Hier sollen alle Befehle, die durch EXTENDED BASIC zusätzlich zur Verfügung gestellt sind, kurz beschrieben werden. Den TI-BASIC Befehlen, die erweitert bzw. verbessert wurden, wenden wir uns später zu.

8.2.1 ACCEPT AT (Z,S)

Dieser Befehl entspricht weitgehend dem INPUT-Befehl und dient zur Eingabe von Daten. Hier besteht jedoch die Möglichkeit, eine Eingabe an jeder beliebigen Bildschirmposition vornehmen zu lassen. Diese Position wird in dem Klammerausdruck durch Z (Zeile) und S (Spalte) spezifiziert. In Verbindung mit diesem Befehl sind viele Optionen möglich, die eine gezielte Dateneingabe erlauben. Nachfolgend wollen wir alle Optionen kurz beschreiben, die für alle Befehle mit dem Zusatz 'AT' gültig sind:

Validate (Bezeichnung)

Es können nur bestimmte Datentypen, die durch den Klammerausdruck festgelegt werden, eingegeben werden. Als Bezeichnung sind folgende Begriffe erlaubt:

Bezeichnung	Bedeutung
DIGIT	Es können nur die Ziffern 0 - 9 eingegeben werden.
UALPHA	Bei der Eingabe sind nur Großbuchstaben gestattet.
'erlaubte Zeichen'	Alle Zeichen, die zwischen den beiden Anführungszeichen stehen, sind bei der Dateneingabe gestattet.

Anmerkung: Durch Komma getrennt, können Sie mehrere Bezeichnungen miteinander verknüpfen.

BEEP

Gibt einen Signalton aus, der darauf aufmerksam macht, daß eine Dateneingabe (bei den Befehlen PRINT AT und DISPLAY AT eine Datenausgabe erfolgt ist.) erwartet wird.

ERASE ALL

Entspricht dem Befehl CALL CLEAR und löscht vor der Datenein/-ausgabe den Bildschirm.

SIZE (Ziffer oder numerische Variable)

Diese Option legt fest, wieviele Zeichen maximal ein- bzw. ausgegeben werden dürfen. Wenn der Wert in der Klammer positiv ist, wird die Zeile, in der die Ein-/Ausgabe erwartet wird, entsprechend dem Klammerwert ab der spezifizierten Bildschirmposition gelöscht. Bei Eingabe eines negativen Wertes bleibt der Bildschirm unverändert, so daß Sie unter Umständen bereits vorhandene Zeichen durch Drücken von >ENTER< übernehmen können.

Wenn Sie SIZE als Option weglassen, wird die Zeile, in der die Eingabe erwartet wird, ab der entsprechenden Spalte komplett gelöscht.

Beispiel: In der Zeile 10 soll ab Spalte 5 die Eingabe einer Zahl vorgenommen werden. Die Zahl darf maximal 8 Zeichen lang sein und kann einen Dezimalpunkt enthalten. Eventuell vorhandene Zahlen können übernommen werden. Ein Ton signalisiert, daß eine Eingabe erwartet wird:

```
10 ACCEPT AT(10,5)BEEP VALIDATE(DIGIT,'.')SIZE(-8):ZAHL
```

Vor Eingabe eines Stringausdruckes in der untersten Zeile ab Spalte 1 soll der Bildschirm gelöscht werden:

```
10 ACCEPT ERASE ALL:Z$
```

Wie Sie sehen, kann der Zusatz AT(Z,S) weggelassen werden. Nun erfolgt die Eingabe wie beim INPUT-Befehl in der 24. Zeile ab Spalte 1.

Wenn Sie ACCEPT AT in TI-BASIC realisieren wollen, muß dies mit den Routinen CALL KEY und CALL HCHAR geschehen, wobei eine Programmierung mit allen Optionen sehr kompliziert ist. Ein kleines Beispiel soll dies verdeutlichen:

Beispiel: In Zeile 10 soll ab Spalte 10 ein String eingegeben werden, der maximal 10 Zeichen beinhalten darf. Ein Ton zeigt die Eingabebereitschaft an:

```
10 CALL CLEAR
20 CALL SOUND(500,261,0)
30 S=10
40 CALL KEY(0,K,ST)
50 IF ST=0 THEN 40
60 IF K=13 THEN 100
70 CALL HCHAR(10,S,K)
80 S=S+1
90 TX$=TX$&CHR$(K)
95 IF LEN(TX$)<10 THEN 40
100 CALL CLEAR
110 PRINT 'IHRE EINGABE:';TX$
120 END
```

Nachdem in Zeile 10 der Bildschirm gelöscht wurde, wird ein Ton (20) ausgegeben, der die Eingabebereitschaft signalisiert. Danach wird in Zeile 30 die Variable S (Spalte) auf Ihren Anfangswert gesetzt. In den Zeilen 40-50 wird auf die Eingabe eines Zeichens gewartet. Wenn Sie >ENTER< eingegeben haben, wird das Programm nach 100 verzweigen (60). Nun folgt über die HCHAR - Routine in Zeile 70 die Ausgabe des eingegebenen Zeichens. Zeile 80 erhöht für das folgende Zeichen die Variable S um 1. In der Variablen TX$ wird Zeichen für Zeichen abgelegt (90). Ist die Gesamtlänge von TX$ gleich 10, wird das Programm zur Ausgabe nach 100 ver-

zweigen.

Sie können anhand dieser simplen Routine leicht erkennen, wie schwierig die Sache wird, wenn noch einige andere Optionen hinzu kommen, und wie leistungsfähig das Modul EXTENDED BASIC ist. Wir wollen Ihnen trotzdem an kleinen Routinen zeigen, wie man die Optionen realisiert. Diese Routinen können Sie dann in Ihre Programme leicht einflechten:

Option VALIDATE:

1. Nur Großbuchstaben

```
10 CALL KEY(0,K,S)
20 IF S=0 THEN 10
30 IF K<65 THEN 10
40 IF K>90 THEN 10
```

2. Nur Ziffern

```
10 CALL KEY(0,K,S)
20 IF S=0 THEN 10
30 IF K<48 THEN 10
40 IF K>57 THEN 10
```

3. Ziffern mit Dezimalpunkt

```
10 CALL KEY(0,K,S)
20 IF S=0 THEN 10
30 IF K=46 THEN 60
40 IF K<48 THEN 10
50 IF K>57 THEN 10
60 (weitere Statements)
```

4. Erlaubte Zeichen

```
10 T$='JN'
20 CALL KEY(0,K,S)
30 IF S=0 THEN 20
40 K$=CHR$(K)
50 FOR I=1 TO LEN(T$)
60 IF K$=SEG$(T$,I,1) THEN 80
70 NEXT I
75 GOTO 20
80 (weitere Statements)
```

Option BEEP:

```
10 CALL SOUND(500,261,0)
```

Option SIZE:

1. Ohne löschen des Eingabefeldes

```
10 LAENGE=5
20 CALL KEY(0,K,S)
30 IF S=0 THEN 20
40 IF K=13 THEN 110
50 Z=Z+1
60 CALL HCHAR(10,10,K)
70 TX$=TX$&CHR$(K)
90 IF LEN(TX$)=LAENGE THEN 110
100 GOTO 20
110 (weitere Statements)
```

2. Mit löschen des Eingabefeldes

```
10 LAENGE=5
20 FOR I=10 TO 15
30 CALL HCHAR(10,I,32)
40 NEXT I
50 CALL KEY(0,K,S)
60 IF S=0 THEN 50
70 IF K=13 THEN 150
80 CALL HCHAR(10,10+Z,K)
90 Z=Z+1
100 TX$=TX$&CHR$(K)
110 IF LEN(TX$)=LAENGE THEN 150
120 GOTO 50
150 (weitere Statements)
```

Nun haben alle TI-BASIC Anwender die Möglichkeit, fast alle Optionen des ACCEPT AT Befehles in ihren Programmen anzuwenden.

8.2.2 CALL CHARPAT (ZC,S$)

Durch diesen Befehl ist es möglich, den Mustercode eines Zeichens, welches im Klammerausdruck mit seinem Zeichencode (ZC) spezifiziert wird, zu bekommen. Der Mustercode wird der Variablen S$ übergeben.

Leider gibt es keinen Weg diesen Befehl für TI-BASIC umzusetzen.

Dieser Befehl ist die Umkehrung der CALL CHAR-Routine. Hier können Sie den Stringausdruck, den man mit CHAR definiert, der Variablen S$ entnehmen.

8.2.3 CALL CHARSET

Diese Funktion bewirkt ein Rücksetzen der Zeichen mit den Codes 32 - 95 in ihren ursprünglichen Zustand. Verwendung findet die Routine dann, wenn Sie den Zeichensatz Ihres TI umdefiniert (CALL CHAR) haben und den Standardsatz benötigen.

In TI-BASIC bleibt Ihnen als einzige Möglichkeit, die Codes in Ihren Ursprungszustand zurückzusetzen, das Drücken der Tasten >FCTN< und >QUIT<. Durch diese beiden Tasten wird ein Systemreset bewirkt, der gleichzeitig alle Daten und Programme die sich im Arbeitsspeicher befinden löscht.

8.2.4 CALL COINC

Wenn Sie selektieren wollen, ob zwei oder alle Sprites miteinander kollidiert sind, ist diese Routine einzusetzen. Die SYNTAX kann folgendermassen aussehen:

CALL COINC (#SPRITE1,#SPRITE2,TOLERANZ,num. Variable)

oder

CALL COINC (#SPRITE,Punktzeile,Punktspalte,Tol.,num.Var.)

oder

CALL COINC (ALL,numerische Variable)

Der numerischen Variablen können Sie entnehmen, ob ein Zusammentreffen stattgefunden hat (-1) oder nicht (0). Näheres entnehmen Sie bitte Ihrem Handbuch zum EXTENDED BASIC.

Bei TI-BASIC besteht keine Möglichkeit, wie bei EXTENDED BASIC Sprites zu erzeugen. Deshalb kann keiner der Befehle, die der Spritebehandlung dienen, für TI-BASIC umgeschrieben werden.

8.2.5 CALL DELSPRITE

Mit CALL DELSPRITE können während des Programmlaufes einzelne oder alle Sprites gelöscht werden.

Beispiel 1: Löschen der SPRITES 1 und 2:

```
CALL DELSPRITE (#1,#2)
```

Beispiel 2: Löschen aller Sprites:

```
CALL DELSPRITE (ALL)
```

Auch hier ist eine Anwendung mit TI-BASIC nicht möglich.

8.2.6 DISPLAY AT (Z,S)

Dieser Befehl ist dem PRINT-Befehl sehr ähnlich, erlaubt jedoch eine Ausgabe an einer beliebigen Bildschirmposition. Auch hier haben Sie die Möglichkeit, durch Optionen die Ausgabedaten zu kontrollieren. Die Optionen sind die gleichen wie unter 8.2.1 beschrieben.

Beispiel 1: Ausgabe von 'DATA BECKER' in der Zeile 10 ab Spalte 8. Zuvor wird der Bildschirm gelöscht:

```
10 DISPLAY AT(10,8)ERASE ALL:'DATA BECKER'
```

Beispiel 2: In der 1. Zeile soll ab Spalte 1 der Inhalt von A$ ausgegeben werden. Die Ausgabe soll durch einen Ton signalisiert werden:

```
10 DISPLAY AT(1,1)BEEP:A$
```

Eine Realisierung dieses Befehles mit TI-BASIC wird auch hier sehr umfangreich sein, wie das folgende Beispiel zeigen wird.

Beispiel: Wie oben in Beispiel 1 wollen wir den String 'DATA BECKER' in der 10. Zeile an Spalte 8 ausge-

ben:

```
10 CALL CLEAR
20 A$='DATA BECKER'
30 FOR I=1 TO LEN(A$)
40 ZC=ASC(SEG$(A$,I,1))
50 CALL HCHAR(10,9+I,ZC)
60 NEXT I
70 GOTO 70
```

Unterbrechen Sie das Programm durch Drücken von >FCTN 4<.

8.2.7 CALL DISTANCE

Mit der Routine DISTANCE können Sie die Entfernung zwischen zwei SPRITES oder einem Sprite und einer Bildschirmposition (Zeile 1 - 256, Spalte 1 - 256) ermitteln. Hierzu die SYNTAX:

CALL DISTANCE (#SPRITE1,#SPRITE2,numerische Variable)

oder

CALL DISTANCE (#SPRITE,Punktzeile,Punktspalte,num.Variable)

Achtung: In der numerischen Variablen wird die Entfernung als Quadratzahl übergeben!

Auch dieser Befehl dient der SPRITE-Behandlung und kann in TI-BASIC nicht realisiert werden.

8.2.8 CALL ERR (FC,FT[,FG,ZN])

Dieser Befehl dient der Fehlerbehandlung und ermittelt den Fehlercode (FC, siehe Anhang) und den Fehlertyp (FT). Wahlweise kann noch das Fehlergewicht (FG, immer 9) und die Zeilennummer (ZN), die gerade abgearbeitet wurde als der Fehler autrat, übergeben werden. Diese Routine wird immer im Zusammenhang mit ON ERROR benutzt (Beisp.siehe unter 8.2.20).

8.2.9 IMAGE

Dieses Statement wird in Verbindung mit PRINT USING benutzt und legt fest, welches Format die auszugebende Zahl haben soll. Der PRINT USING Befehl bezieht sich dann auf die Zeilennummer, in der IMAGE steht.

Beispiel: Wir legen fest, daß eine Zahl mit drei Vorkomma- und zwei Nachkommastellen ausgegeben wird und lassen die Zahl 134.6356 mit dem PRINT USING Befehl darstellen:

```
10 CALL CLEAR
20 IMAGE ###.##
30 PRINT USING 20:134.6356
40 END
RUN
134.64

READY.
```

8.2.10 CALL INIT

Im Zusammenhang mit LINK, LOAD und PEEK wird diese Routine angewandt, um Assembler-Programme zu benutzen. INIT überprüft den Rechner, ob alle Voraussetzungen zur Benutzung von Assembler-Programmen erfüllt sind.

Um diesen Befehl benutzen zu können, brauchen Sie eine Speichererweiterung.

8.2.11 CALL LINK

Dieser Befehl übergibt Daten an ein Unterprogramm in Assemblersprache. Im übrigen gilt das gleiche wie bei CALL INIT!

8.2.12 LINPUT

Dieser Befehl ist dem INPUT-Befehl sehr ähnlich, nur können hier alle Separatoren wie Kommata, Semikolon oder Doppelpunkt mit eingegeben werden. Dies gilt sowohl bei der normalen Bildschirmeingabe als auch bei der Datenbehandlung mit externen Dateien, wo zusätzlich eine File-Nummer (#) angegeben werden muß.

Mit TI-BASIC können Sie diesen Befehl, wie unter 8.2.1 beschrieben, mit CALL KEY leicht simulieren.

8.2.13 CALL LOAD

CALL LOAD wird in Verbindung mit den Routinen INIT, LINK und PEEK benutzt, um Assemblerprogramme anzuwenden.

CALL LOAD dient zum Datentransfer von Cassette oder Diskette in die Speichererweiterung. Diese Daten werden in der Speichererweiterung als sogenannte Direkt-Objekt-Daten abgelegt. Man kann diesen Befehl auch mit dem POKE Befehl vergleichen. In einem Klammerausdruck müssen Parameter übergeben werden, mit denen eine bestimmte Adresse im Speicher angesprochen und belegt wird.

Beispiel: Wir wollen einmal den bekannten POKE-Befehl der CALL LOAD - Routine mit einem kleinen Anwendungsbeispiel gegenüberstellen.

```
POKE 22122,25
POKE 22123,65
POKE 22124,27
```

Diese Funktion erfüllt den gleichen Zweck:

```
CALL LOAD(22122,25,65,27)
```

Mit diesem Befehl müssen Sie besonders vorsichtig umgehen, da sich bei falscher Anwendung der Rechner sperren kann. Dieser Zustand ist nur durch Aus- und Wiedereinschalten zu unterbrechen, was den Verlust aller Daten im RAM zur Folge hat!!

8.2.14 CALL LOCATE

Mit LOCATE können Sie ein oder mehrere SPRITES an einer beliebigen Bildschirmposition positionieren. Die SYNTAX:

CALL LOCATE (#Spritenummer,Punktzeile,Punktspalte[,.....])

Dieser Befehl ist für TI-BASIC nicht zu emulieren.

8.2.15 CALL MAGNIFY

Hier können Sie die Größe eines Zeichens oder Sprites beeinflussen. Ferner legen Sie mit MAGNIFY fest, aus wievielen Zeichen ein Sprite besteht (max.4). Die SYNTAX:

CALL MAGNIFY (Vergrößerungsfaktor)

Vergrößerungsfaktoren:

1 -- Ein Sprite einfacher Grüße, bestehend aus einem Zeichen.
2 -- Sprite in doppelter Größe, bestehend aus einem Zeichen.
3 -- Sprite in einfacher Größe, bestehend aus vier Zeichen.
4 -- Sprite in doppelter Größe, bestehend aus vier Zeichen.

Bei Sprites, die aus vier Zeichen bestehen, sind die Codes der betreffenden Zeichen vier aufeinanderfolgende Zahlen, die wie folgt angeordnet sein müssen:

1	3
2	4

8.2.16 MAX (Wert1,Wert2)

Diese Funktion gibt an, welcher der beiden, im Klammerausdruck stehenden Werte, der größere ist.

Beispiel: A=MAX(B,C)

In TI-BASIC kann man diese Funktion folgendermaßen simulieren:

```
10 IF B>C THEN 11 ELSE 20
11 A=B
12 GOTO 40
20 IF C>B THEN 21 ELSE 30
21 A=C
22 GOTO 40
30 IF B=C THEN 21 ELSE 40
31 A=B
40 PRINT A
50 END
```

8.2.17 MERGE

Dieser Befehl dient zum Anhängen eines BASIC-Programmes an ein schon im Arbeitsspeicher befindliches BASIC-Programm. Beachten Sie hierbei, daß alle gleichen Zeilennummern von dem nachgeladenen Programm überschrieben werden!

Man kann diesen Befehl nur in Verbindung mit einer Diskettenstation anwenden.

Besonders nützlich ist die Fähigkeit dieser Routine, Unterprogramme, die oft benutzt werden, in ein bestehendes Programm einzubetten. Das Unterprogramm muß allerdings mit dem Zusatz 'MERGE' auf Diskette gespeichert sein.

Beispiel: Der folgende Befehl lädt ein BASIC-Programm an ein, schon im Arbeitsspeicher befindliches Programm nach.

```
MERGE DSK 1.QUICKSORT
```

8.2.18 MIN (Wert1,Wert2)

Die MIN-Funktion ist die Umkehrung von MAX. Hier wird bestimmt, welcher der beiden Werte der kleinere ist.

Beispiel: A=MIN(B,C)

Wenn Sie diese Funktion in TI-BASIC programmieren wollen, verwenden Sie bitte unser Beispiel aus 8.2.16. Sie müssen in

den Zeilen 10 und 20 lediglich die Größerzeichen (>) in Kleinerzeichen (<) umwandeln. Hier sind die Zeilen, die Sie dort einsetzen können:

```
10 IF B<C THEN 11 ELSE 20
20 IF C<B THEN 21 ELSE 30
```

8.2.19 CALL MOTION

Mit dieser Routine kann die Geschwindigkeit eines Sprites festgelegt werden. Ferner können Sie durch Eingabe der Zeilen- und Spaltengeschwindigkeit die Bewegungsrichtung eines Sprites beeinflußen. Die SYNTAX:

```
CALL MOTION (#Spritenummer,Zeilengeschw.,Spaltengeschw.)
```

Die Geschwindigkeitswerte können zwischen -128 und +127 liegen, wobei das Vorzeichen die Bewegungsrichtung bestimmen soll.

8.2.20 ON ERROR

Mit Hilfe dieser Routine können Sie dem Programm mitteilen, was geschehen soll, wenn ein Fehler auftritt. Die SYNTAX:

```
ON ERROR STOP
    ODER
ON ERROR Zeilennummer
```

Beispiel:

```
10 REM ** FEHLERBEHANDLUNG **
20 CALL CLEAR
30 ON ERROR 200
40 PINT 'HIER IST DER FEHLER'
50 END
200 ** FEHLERERKENNUNG **
210 CALL ERR(FC,FT,FG,ZN)
220 IF FC<>14 THEN RETURN 30
230 PRINT 'SYNTAX ERROR IN';ZN
240 STOP
```

8.2.21 ON WARNING

Hiermit können Sie bestimmen, was geschehen soll, wenn in einem Programm ein leichter Fehler auftritt. Unter leichten Fehlern versteht man Störungen, bei denen ein Programm nicht unterbrochen wird, sondern mit Ausgabe einer Warnung weiterläuft. Die SYNTAX:

```
ON WARNING PRINT
     oder
ON WARNING STOP
     oder
ON WARNING NEXT
```

Bei ON WARNING NEXT wird der Programmzeiger einfach - ohne Fehlermeldung - zur nächsten Zeile übergehen.

Beispiel:

```
10 REM ** ON WARNING **
20 ON WARNING NEXT
30 X=18/0
40 ON WARNING PRINT
```

```
50 PRINT 9/0
60 ON WARNING STOP
70 A=125/0
80 (weitere Statements)
```

Dieses kleine Programm demonstriert sehr wirkungsvoll die 3 Möglichkeiten des ON WARNING-Befehles. Der Fehler in Zeile 30 wird einfach ignoriert und das Programm macht mit Zeile 40 weiter. Beim Fehler in Zeile 50 wird auf dem Bildschirm eine Warnung (WARNING:DIVISION BY ZERO) ausgegeben und der Fehler in Zeile 70 führt schließlich zum Abbruch der Routine.

8.2.22 CALL PATTERN

Wenn Sie mitten in einem Programm die Form eines Sprites verändern wollen, können Sie dies mit diesem Befehl tun. Es ist auch möglich mehrere Sprites mit einem PATTERN-Befehl zu verändern. Durch CALL PATTERN wird zwar die Form eines Sprites verändert, die Eigenschaften bleiben jedoch weiterhin erhalten. Die SYNTAX:

```
CALL PATTERN (#Spritenummer,Zeichencode[,.....])
```

Das angegebene Sprite wird nun die Form annehmen, die das Zeichen mit dem angegebenen Zeichencode besitzt.

Beispiel: Mit dem folgenden Befehl kann man Sprite 1 in ein Sternchen verwandeln:

```
CALL PATTERN(#1,42)
```

8.2.23 CALL PEEK

CALL PEEK wird zusammen mit INIT, LOAD und LINK verwendet, um Assenblerprogramme benutzen zu können.

Mit dieser Funktion können Speicherzellen im Arbeitsspeicher des TI einzeln gelesen werden (ab Adresse 16383).

Dieser Befehl kann ohne Assemblerprogramm verwendet werden, was jedoch nicht empfehlenswert ist, da hier das Programm undefiniert verzweigen kann.

8.2.24 PI

Immer wenn Sie die Variable aufrufen, steht der genaue Wert PI (3.14159265359) zur Verfügung.

Beispiel: Berechnung des Umfanges eines Kreises mit einem Durchmesser von 5 Metern.

EXTENDED BASIC:

```
10 D=5
20 U=((D*D)*PI)/4
30 PRINT U
```

TI-BASIC:

```
10 PI=(4*ATN(1))
20 U=((5*5)*PI)/4
30 PRINT U
```

8.2.25 CALL POSITION

CALL POSITION stellt die momentane Position eines oder mehrerer Sprites fest. Die SYNTAX:

CALL POSITION (#Spritenummer,Punktzeile,Punktspalte[,...])

Den Variablen für Punktspalte und Punktzeile werden die jeweiligen Positionswerte (1 - 256) der Sprites zugewiesen.

Diese Routine findet besonders dort Anwendung, wo man in selbstprogrammierten Spielen erreichen will, daß sich zwei Sprites gegenseitig bekämpfen. Man ermittelt einfach die Position des einen Sprites und teilt sie dem anderen mit.

8.2.26 PRINT USING

Dieser Befehl wurde in diesem Buch schon mehrfach besprochen und dient der formatierten Ausgabe von Zahlen. Der benötigte Formatstring kann entweder direkt folgen oder in einem IMAGE Statement festgelegt sein (siehe auch 8.2.9). Die SYNTAX:

```
PRINT USING '###.##':A
      oder
PRINT USING 100:A
```

Im letzteren Fall muß in der Zeile 100 ein IMAGE-Statement stehen, daß den Formatstring (###.##) beinhaltet.

Für die Realisierung dieses Befehles sehen Sie bitte im Kapitel 'Tip's und Trick's' nach. Dort ist eine Möglichkeit

ausführlich beschrieben (gemeint ist hier die Realisierung in TI-BASIC).

8.2.27 REC

Diese Funktion gibt an, welche Datensatznummer mit den Befehlen PRINT, INPUT oder LINPUT in einer Datei als nächste angesprochen wird. Die SYNTAX:

```
PRINT REC(2)
INPUT REC(2)
LINPUT REC(2)
```

8.2.28 RPT$

Diese Funktion liefert einen String, der in dem Klammerausdruck definiert wird. Die SYNTAX:

```
RPT$(String,numerischer Ausdruck)
```

Der String legt fest, aus welchen Zeichen der komplette Stringausdruck bestehen soll. Im numerischen Ausdruck wird angegeben, wie oft die Zeichen zu wiederholen sind.

Beispiel: Wir wollen der Variablen A$ eine Zeichenkette, bestehend aus 20 Sternchen (*) zuweisen. Mit der Funktion RPT$ ist dies sehr einfach:

```
10 A$=RPT$('*',20)
20 PRINT A$
```

In TI-BASIC müßte man so programmieren:

```
10 A$='********************'
20 PRINT A$
```

8.2.29 CALL SAY

Dieser Befehl ist nur in Verbindung mit einem Sprachsyntesizer sinnvoll. Eine genaue Beschreibung der Routine entnehmen Sie bitte dem entsprechenden Handbuch.

8.2.30 SIZE

Mit dieser Funktion können Sie den freien Speicherplatz im Arbeitsspeicher feststellen.

In TI-BASIC können Sie das gleiche mit der folgenden kleinen Routine erreichen, die schon im Kapitel 'Tip's und Trick's' beschrieben wurde:

```
1 RAM=RAM+7
2 GOSUB 1
PRINT RAM
```

8.2.31 CALL SPGET

Wie bei CALL SAY ist hier eine Anwendung nur Sprachsyntesizer möglich. Auch hier entnehmen Sie bitte die genaue Beschreibung dem Handbuch zum Sprachsyntesizer.

8.2.32 CALL SPRITE

Mit CALL SPRITE werden die einzelnen Sprites definiert, die Sie in Ihren Programmen verwenden wollen. Die SYNTAX:

CALL SPRITE(#Spritenummer,ZC,FC,PZ,PS[,ZG,SG])

Die Parameter haben folgende Bedeutung:

#Spritenummer -- Muß ein Wert zwischen 1 und 28 sein und legt die Nummer fest, unter der der Sprite später aufgerufen wird.

ZC -- Zeichencode, der das Muster des Sprites enthält.

FC -- Farbcode. Gibt an, in welcher Farbe der Sprite dargestellt wird.

PZ -- Punktzeile, wo der Sprite positioniert wird

PS -- Punktspalte der Spriteposition.

ZG -- Hier kann wahlweise die Zeilengeschwindigkeit eingegeben werden.

SG -- Wahlweise Eingabe der Spaltengeschwindigkeit.

Ein Programm, das den Einsatz von Sprites demonstriert, finden Sie im Kapitel 3 (MINENFELD).

8.2.33 SUB

Dieser Befehl kennzeichnet ein selbsterstelltes Unterprogramm, das im Programmverlauf mit CALL Name aufgerufen werden kann. Die SYNTAX:

SUB Name

Als Namen können Sie jeden beliebigen Stringausdruck eingeben, unter dem die Routine später mit dem Zusatz 'CALL' aufgerufen werden kann.

Beispiel:

```
10 CALL CLEAR
20 PRINT 'ANWEISUNGEN (J/N) ?'
30 ACCEPT AT(24,1)VALIDATE('JN'):Q$
40 IF Q$='N' THEN 60
50 CALL ANWEISUNG
60 CALL CLEAR
      .
      .
      .
10000 SUB ANWEISUNG
      .
      .
10100 IF X=A THEN SUBEXIT
      .
      .
10200 SUBEND
```

Alle selbstdefinierten Unterprogramme müssen am Ende eines Hauptprogrammes plaziert werden. Im obigen Beispiel wird der Programmzeiger bei Erreichen von SUBEND oder SUBEXIT zur Programmzeile 60 zurückkehren.

8.2.34 SUBEND

Diesen Befehl können Sie im weitesten Sinne mit RETURN vergleichen. Er kennzeichnet das Ende eines Unterprogrammes.

Beispiel: siehe 8.2.33

8.2.35 SUBEXIT

Dieser Befehl erlaubt das vorzeitige Verlassen eines Unterprogrammes und wird meist bei Vergleichsoperationen verwandt.

Beispiel: siehe 8.2.33

8.2.36 CALL VERSION

Mit dieser Routine können Sie feststellen, welche BASIC-Version gegenwärtig aktuell ist. Die SYNTAX:

CALL VERSION (numerische Variable)

Der numerischen Variablen wird zum Beispiel bei Verwendung von EXTENDED BASIC der Wert 110 zugewiesen.

Die vorangegangenen Befehlsbeschreibungen sind als kurzer, oberflächlicher Überblick über die Möglichkeiten des EXTENDED BASIC Moduls zu verstehen. Wenn Sie mehr über die Fähigkeiten und die einzelnen Befehle wissen wollen, können wir Ihnen folgendes Buch empfehlen, das in deutsch abgefasst ist:

TI BASIC/EXTENDED BASIC für Anfänger und Fortgeschrittene
Erschienen bei: Texas Instruments Deutschland GmbH
Learning Center
Haggertystrasse 1
8050 Freising

8.3 Verbesserte Befehle des TI-BASIC

Wie an anderer Stelle schon erwähnt, sind einige Befehle des TI-BASIC erweitert und verbessert worden, bzw. haben zusätzliche Funktionen bekommen. Diese Befehle wollen wir im Folgenden kurz aufzeigen.

8.3.1 CALL CHAR

Mit dieser Routine werden jetzt zusätzlich die Formen der Sprites definiert. Beim Entwurf der Muster, gehen Sie bitte genauso vor, wie unter 6.4 beschrieben.

8.3.2 CALL COLOR

CALL COLOR definiert im EXTENDED BASIC zusätzlich die Farben der Sprites. Die SYNTAX:

CALL COLOR (#Spritenummer,Vordergrundfarbe[,.....])

8.3.3 DIM

Während Sie mit diesem Befehl in TI-BASIC 3-dimensionale Felder definieren konnten, können mit EXTENDED BASIC 7-dimensionale Felder aufbauen.

8.3.4 EDIT

Zum Editieren von Programmen muß man im TI-BASIC folgende SYNTAX eingeben:

EDIT Zeilennummer >ENTER<

Im EXTENDED BASIC hat man die Editiermöglichkeiten etwas verbessert, indem die Eingabe verkürtzt wurde. Zum Aufrufen einer Zeilennummer im Edit-Modus haben Sie jetzt folgende Möglichkeiten:

Zeilennummer >FCTN E<
oder
Zeilennummer >FCTN X<

Wenn Sie sich bereits im Edit-Modus befinden, können Sie durch Drücken von >FCTN E< jeweils die nächst höhere Zeilennummer im Programm anzeigen und mit >FCTN X< die nächst niedrigere.

8.3.5 IF - THEN - ELSE

Bisher konnten nach THEN bzw. ELSE nur Zeilennummern folgen, während bei EXTENDED BASIC auch Befehlslisten folgen können. Dies ist eine der leistungsfähigsten Verbesserungen, die das Modul mit sich bringt. An kurzen Beispielen soll das verdeutlicht werden:

Beispiel 1: IF X<=B THEN PRINT A$ :: GOTO 100
Beispiel 2: IF X=6 THEN 100 ELSE END

8.3.6 RUN

Der RUN-Befehl kann, im Gegensatz zu TI-BASIC,bei EXTENDED BASIC auch als Statement, also in einer Zeilennummer auftauchen.

Beispiel:
```
100 PRINT 'NOCHMAL (J/N) ?'
110 CALL KEY(0,K,S)
120 IF S=0 THEN 110
130 IF CHR$(K)='N' THEN END
140 IF CHR$(K)='J' THEN RUN
150 GOTO 110
```

Wir haben nun die wesentlichen Merkmale des EXTENDED BASIC Moduls beschrieben. Sicherlich haben diejenigen unter Ihnen, die über diese BASIC-Erweiterung nicht verfügen, einen ausreichenden Einblick über die Möglichkeiten und erweiterten Graphikfähigkeiten dieses Moduls erhalten. Vielleicht konnte dieses Kapitel den Zweiflern, die sich noch nicht ganz einig waren, ob dieses Modul angeschafft werden soll, eine kleine Entscheidungshilfe geben. Wer professionelle und leistungsfähige Programme schreiben will, kommt einfach nicht um die Anschaffung dieses Modules herum.

KAPITEL 9: EIN PROGRAMM ENTSTEHT

Wenn komplexe Programme entwickelt werden sollen, ist es nicht mehr möglich, nach alter 'Hacker-Manier' solch ein Programm direkt am Bildschirm zu schreiben. Eine umfangreiche Schreibtischarbeit geht der Bildschirmarbeit voraus. Viele Hobby-Programmierer gehen den falschen Weg und schreiben Ihre Programmdokumentation, nachdem das Programm steht. Solche Programme sind meistens nicht von hoher Qualität, da die intensive Vorbereitung und Planung fehlt. Wer sein Programm am Schreibtisch entwickelt, verliert nie den Überblick und kann die meisten Fehlerquellen von vornherein ausschließen. In diesem Kapitel möchten wir Ihnen einen Leitfaden an die Hand geben, nachdem sich auch die meisten Profi's orientieren.

Ein kleines Beispiel, das diese Problematik illustrieren soll, wird Sie das ganze Kapitel hindurch begleiten. Ein kleines Programm, das die Werte (Winkel,Schenkel) eines rechtwinkligen Dreieckes berechnet, soll unserem Beispiel als Grundlage dienen. Am Ende des Kapitels werden Sie dieses Programm mit einer vollständigen Dokumentation in Händen halten.

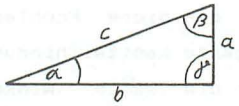
9.1 Definition der Aufgabe

Zuerst sollten Sie einmal niederschreiben, was das Programm eigentlich können muß. Vor allem das 'WAS PASSIERT WENN' sollte eingehend analysiert werden. Tragen Sie alle mathematischen Formeln und Problemlösungen zusammen und überlegen Sie sich, wieviel Speicherplatz ungefähr benötigt wird. Finden Sie heraus, welche Programmteile, die sich immer wieder wiederholen, als Unterprogramme entwickelt werden können. Wenn Sie all diese Dinge erledigt haben, können Sie zum nächsten Schritt übergehen.

Beispiel:

Programm zur Berechnung eines rechtwinkligen Dreieckes

Die Winkel und Schenkel werden wie folgt bezeichnet:



Das Programm soll nach Eingabe von zwei Werten, die zuvor in einer Menuauswahl festgelegt werden, alle Berechnungen für das Dreieck durchführen. Durch eine Überprüfung der Eingabedaten soll verhindert werden, daß der Rechner abstürzt. In diesem Fall wird eine Neueingabe eingeleitet.

Formeln:

$$\sin\alpha = \frac{\text{Gegenkathete}}{\text{Hypotenuse}} = \frac{a}{c} = \cos\beta$$

$$\cos\alpha = \frac{\text{Ankathete}}{\text{Hypotenuse}} = \frac{b}{c} = \sin\beta$$

$$\tan\alpha = \frac{\text{Gegenkathete}}{\text{Ankathete}} = \frac{a}{b} = \cot\beta$$

$$\cot\alpha = \frac{\text{Ankathete}}{\text{Gegenkathete}} = \frac{b}{a} = \tan\beta$$

aus diesen Formeln lassen sich alle benötigten Berechnungen ableiten.

Alle Formeln werden als Unterprogrammroutinen programmiert.

Die Speicherkapazität des Programmes wird nicht über 2kB liegen.

9.2 Festlegen der Variablen

In Ihrem zukünftigen Programm werden sicherlich Daten unterschiedlichster Art verarbeitet. Vergeben Sie daher schon jetzt alle Variablennamen, die Sie in Ihrem Programm verwenden wollen (wem ist es nicht schon einmal passiert, daß er die Variablennamen bei der Eingabe eines Programmes vergessen hat?). Ordnen Sie auf Papier gleichzeitig die Anfangswerte den definierten Variablen zu.

Beispiel:

Festlegen der Variablen:

Gegenkathete	= GK	cot β	= C2
Ankathete	= AK	tan β	= TB
Hypotenuse	= HY	sin γ	= CA
Alpha	= A	cos γ	= SA
Beta	= B	tan γ	= C1
Gamma	= G	cot γ	= TA
sin α	= SA	Für evtl. benötigte Zähl- und Statusregister werden folgende Variablen vorgesehen:	
cos α	= CA		
tan α	= TA	I	L
cot α	= C1	J	M
cos β	= CB	K	N
sin β	= SB		

9.3 Formulieren der mathematischen Probleme

Jetzt sollten Sie alle mathematischen Formeln, die Sie schon zusammengetragen haben, in die BASIC-Sprache umsetzen und auf dem Computer kurz austesten. Auf diese Art und Weise können Sie eine, oft kritische Fehlerquelle ausschalten. Alle Formeln und Werte sollten hier, vor dem ersten Programmlauf 'zu Fuß' ausgerechnet werden, damit eine Vergleichsmöglichkeit beim Testen der Programme vorhanden ist.

Beispiel:

Umsetzen der Formeln:

1. SA = GK/HY = CB (Berechnung sinα)
   GK = SA * HY
   HY = GK/SA

2. CA = AK/HY = SB (Berechnung cosα)
   AK = CA * HY
   HY = AK/CA

3. TA = GK/AK = CB (Berechnung tanα)
   GK = TA * AK
   AK = GK/TA

4. C1 = AK/GK = C2 (Berechnung cotα)
   AK = C1 * GK
   GK = AK/C1

Der Testlauf zeigte, daß alle Formeln stimmen.

9.4 Festlegen der Programmstruktur

Nun halten Sie schriftlich fest, in welcher Reihenfolge das Programm die gestellten Probleme abarbeitet. Dies soll für den nächsten Schritt - das Flußdiagramm - als Grundlage dienen. Arbeiten Sie hier besonders sorgfältig, da in der Regel bei diesem Schritt die meisten Fehler gemacht werden.

Beispiel:

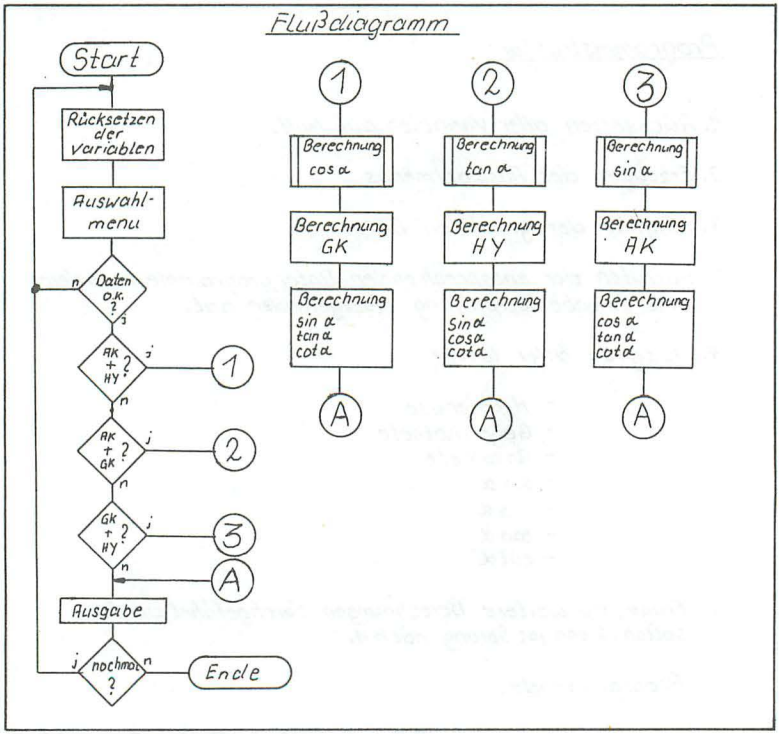
Programmstruktur:

1. *Rücksetzen aller Variablen auf Null.*
2. *Erzeugen des Auswahlmenus.*
3. *Eingabe der gewählten Daten.*
4. *Aufrufen der entsprechenden Unterprogramme nachdem eine Eingabeüberprüfung stattgefunden hat.*
5. *Ausgabe aller Werte:*
   - *Hypotenuse*
   - *Gegenkathete*
   - *Ankathete*
   - *sin α*
   - *cos α*
   - *tan α*
   - *cot α*
6. *Frage, ob weitere Berechnungen durchgeführt werden sollen. Wenn ja: Sprung nach 1.*
7. *Programmende*

9.5 Das Flußdiagramm

Das Flußdiagramm ist der letzte Schritt in der Programmentwicklung. Mit Hilfe von diesen Datenflußplänen können Sie jedes Problem auf Papier lösen. Vor allem können fast alle logischen Fehler, die auftreten könnten, durch dieses Flußdiagramm entdeckt werden. Wer ein Flußdiagramm erstellen will, muß einige Zeichen, die man in solchen Diagrammen verwendet, kennen. Sie können all diese Zeichen in der einschlägigen Literatur finden. Wir gehen davon aus, daß die Mehrzahl von Ihnen diese Zeichen schon kennt und verzichten deshalb auf eine ausführliche Erklärung dieser Symbole.

Beispiel:



9.6 Das Programm als Listing

Erstellen Sie nun da Programm Zeile für Zeile im BASIC-Code. Wenn Sie die letzte Zeilennummer aufgeschrieben haben, liegt das komplette Listing vor Ihnen und das Programm kann in den Computer eingegeben werden. Wenn das Programm vollständig eingegeben ist, sollte es mit mehreren Testläufen ausprobiert werden.

Sollte sich bei den Testläufen herausstellen, daß im Programm ein Fehler ist, so haben Sie bereits durch die gründliche Vorbereitung alle Mittel in der Hand, um diesen Lapsus schnell zu finden.

Beispiel: Wir zeigen Ihnen hier das komplette Listing unseres Beispielprogrammes:

```
10 CALL CLEAR
20 GK=0
30 HY=0
40 AK=0
50 SA=0
60 CA=0
70 TA=0
80 C1=0
90 CB=0
100 SB=0
110 C2=0
120 TB=0
130 CA=0
140 SA=0
150 C1=0
160 TA=0
```

```
170 PRINT 'BITTE KENNZIFFER EINGEBEN!'
180 PRINT ::::'[1] - ANKATHETE UND'
190 PRINT '        HYPOTENUSE'
200 PRINT ::'[2] - ANKATHETE UND'
210 PRINT '      GEGENKATHETE'
220 PRINT ::'[3] - GEGENKATHETE UND'
230 PRINT '     HYPOTENUSE'
240 PRINT ::::
250 CALL KEY (0,K,S)
260 IF S=0 THEN 250
270 IF K<49 THEN 250
280 IF K>51 THEN 250
290 IF K-48<>1 THEN 340
300 CALL CLEAR
310 INPUT 'ANKATHETE:':AK
320 INPUT 'HYPOTENUSE:':HY
330 GOTO 1000
340 IF K-48<>2 THEN 390
350 CALL CLEAR
360 INPUT 'ANKATHETE:':AK
370 INPUT 'GEGENKATHETE:':GK
380 GOTO 1500
390 CALL CLEAR
400 INPUT 'GEGENKATHETE:':GK
410 INPUT 'HYPOTENUSE:':HY
420 GOTO 2000
430 CALL CLEAR
440 PRINT 'HYPOTENUSE  =';HY
450 PRINT 'ANKATHETE   =';AK
460 PRINT 'GEGENKATHETE=';GK
470 PRINT 'SIN ALPHA   =';SA
480 PRINT 'COS ALPHA   =';CA
490 PRINT 'TAN ALPHA   =';TA
500 PRINT 'COT ALPHA   =';C1
510 PRINT ::::'NOCHMAL (J/N) ?'
```

```
520 CALL KEY(0,K,S)
530 IF S=0 THEN 520
540 IF CHR$(K)='J' THEN 10
550 END
1000 CA=AK/HY
1010 GK=SQR((HY*HY)-(AK*AK))
1020 SA=GK/HY
1030 TA=GK/AK
1040 C1=AK/GK
1050 GOTO 430
1500 TA=GK/AK
1510 HY=SQR((GK*GK)+(AK*AK))
1520 SA=GK/HY
1530 CA=AK/HY
1540 GOTO 1040
2000 SA=GK/HY
2010 AK=SQR((HY*HY)-(GK*GK))
2020 CA=AK/HY
2030 GOTO 1030
```

9.7 Die Dokumentation

Wenn Ihr Programm endlich läuft, können Sie alle Unterlagen, die Sie bisher erarbeitet haben korrigieren und zusammenfassen. Erstellen Sie sich jetzt ein zusätzliches Blatt, auf dem Programmiertag (Datum) und Name des Programmierers enthalten ist und fügen eine kleine Programmbeschreibung bei, damit auch fremde Anwender Ihr Programm benutzen können. Sie haben jetzt eine vollständige Dokumentation Ihres Programmes, die folgende Abschnitte enthalten sollte:

1. Definition der Aufgabe
2. Variablenliste

3. Mathematische Formeln
4. Programmstruktur
5. Flußdiagramm
6. Listing
7. Programmbeschreibung

Einen Punkt haben wir bei der Erstellung unseres Beispieles unterschlagen: Die Programmbeschreibung. Das Programm ist jedoch so einfach und simpel gestaltet, daß wir uns diesen letzten Schritt getrost sparen können. Sie werden längst erkannt haben, daß man aus unserem Beispielprogramm erheblich mehr machen könnte, doch wurde es aus Demonstrationszwecken absichtlich so bieder gehalten.

Achten Sie darauf, daß in Ihrem Proramm alle möglichen Eingabefehler berücksichtigt sind. Denn:

** EIN GUTES PROGRAMM STÜRZT NICHT AB **

KAPITEL 10: SPRITES UND IHRE ANWENDUNG

In diesem Buch haben Sie den Begriff 'SPRITE' schon an mehreren Stellen gelesen. In unserem letzten Kapitel wollen wir Sie kurz in die Gestaltung und Anwendung dieser Graphikkobolde einweisen. Doch bevor wir ins Eingemachte gehen, soll noch eine Frage geklärt werden:

Was sind eigentlich Sprites ?

Sprites sind frei definierbare Graphikfiguren oder -symbole, die sich unabhängig vom übrigen Bildschirmgeschehen in einem Programm verwenden lassen. Sicherlich haben Sie alle schon einen Sprite gesehen, denn in Telespielen werden vornehmlich Graphikfiguren dieser Sparte verwendet (z.B. PAC MAN oder SPACE INVADER). Sie können Sprites auf einem Bildschirm in alle möglichen Richtungen (ruckfrei) steuern oder feststellen, ob zwei Sprites zusammengestoßen sind.

10.1 Wie entstehen Sprites ?

Sprites werden ähnlich wie Sonderzeichen definiert (siehe Kapitel 6.4). Sie können aber auch jedes andere Zeichen, das auf Ihrem Rechner zur Verfügung steht zum Sprite 'erklären'. Bis zu 28 Sprites können Sie erstellen und gleichzeitig auf dem Bildschirm darstellen. Alle Sprites werden in dem Speicherplatz gespeichert, der vorher für die ASCII-Codes 144 - 159 reserviert war. Wir wollen jetzt einmal einen Sprite erstellen.

Bei der Erstellung des Sprites setzen wir die Kenntnis des CALL CHAR - Befehles und seiner Anwendung voraus. Sollte Ihnen dieser Befehl noch unbekannt sein, dann schlagen Sie jetzt das Kapitel 6.4 (CALL CHAR) auf und lesen das dort vermittelte einmal durch.

Unser Sprite, den wir jetzt definieren wollen, soll die Form eines Rennwagens haben. Wir nehmen uns zuerst wieder ein kariertes Blatt Papier und teilen es in ein Raster mit 8 x 8 Kästchen auf. Nun erstellen wir in der bekannten Form den entsprechenden Musterstring:

```
* . . *:* . . *   =>   1 0 0 1:1 0 0 1   =>   9 : 9
* * * *:* * * *   =>   1 1 1 1:1 1 1 1   =>   F : F
* . * *:* * . *   =>   1 0 1 1:1 1 0 1   =>   B : D
. . * *:* * . .   =>   0 0 1 1:1 1 0 0   =>   3 : C
. . * .:. * . .   =>   0 0 1 0:0 1 0 0   =>   2 : 4
* . * *:* * . *   =>   1 0 1 1:1 1 0 1   =>   B : D
* * * *:* * * *   =>   1 1 1 1:1 1 1 1   =>   F : F
* . * *:* * . *   =>   1 0 1 1:1 1 0 1   =>   B : D
```

Mit CALL CHAR definieren wir den ASCII-Code 128 mit dem oben erstellten Musterstring ('99FFBD3C24BDFFBD'):

```
CALL CHAR (128,'99FFBD3C24BDFFBD')
```

Wann immer wir diesen 'Rennwagen' brauchen, können wir ihn unter seinem ASCII-Code aufrufen. Wir haben jetzt ein Zeichen mit der Form eines Rennwagens definiert. Einen Sprite machen wir jetzt aus ihm.

Im Kapitel 8 haben wir den Befehl CALL SPRITE schon einmal kurz vorgestellt. Mit diesem Befehl können wir den Sprite - unseren Rennwagen - ins Leben rufen und Ihn an einer Bildschirmposition darstellen. Bei der Positionierung von Sprites müßen Sie beachten, daß sich der Bildschirm nun in 256 Punktspalten und 256 Punktzeilen untergliedert. In einer stark gerafften Darstellung wollen wir Ihnen dies veranschaulichen:

```
      1--------Punktspalten------256
    1 ............................
    I ............................
    P ............................
    u ............................
    n ............................
    k ............................
    t ............................
    z ............................
    e ............................
    i ............................
    l ............................
    e ............................
    I ............................
  256 ............................
```

Zu der obigen Darstellung muß angemerkt werden, daß die Punktzeilen 192 - 256 im unsichtbaren Bereich liegen. Sie können zwar Ihre Sprites dort positionieren, jedoch nicht visuell wahrnehmen (nicht sehen).

Wenn wir nun unser Sprite positionieren, geben wir durch Punktzeile und Punktspalte die linke obere Ecke des Sprites an.

Unser Rennwagen soll die Spritenummer 1 erhalten und in der Mitte des Bildschirmes positioniert werden:

```
CALL SPRITE (#1,128,7,90,120)
```

Der Sprite wir ungefähr in der Bildschirmmitte positioniert!

Nachdem Sie nun mit Entwurf und Darstellung eines Sprites vertraut sind, soll es nun in Bewegung versetzt werden.

10.2 Ein Sprite bewegt sich

Um einen Sprite in Bewegung zu versetzen dient die MOTION Routine, die mit dem Zusatz 'CALL' aufgerufen wird. Die Bewegungsrichtung und die Geschwindigkeit wird durch Eingabe einer Spalten - und Zeilengeschwindigkeit festgelegt. Beide Angaben können einen Wert zwischen -128 und +127 annehmen, wobei das Vorzeichen die Richtung bestimmt. Sind beide Werte '0', wird der Sprite stillstehen.

Beispiel: Unser Rennwagen wird in der Bildschirmmitte dargestellt und in Bewegung versetzt. Der Sprite wird sich solange bewegen, bis Sie Ihn mit den Tasten >FCTN 4< zum Anhalten zwingen:

```
10 CALL CLEAR
20 CALL CHAR(128,'99FFBD3C24BDFFBD')
30 CALL SPRITE(#1,128,7,90,120)
40 CALL MOTION(#1,50,0)
50 GOTO 50
```

Bevor Sie die Bewegung der Sprites vollkommen kontollieren können, müssen Sie sicherlich noch ein wenig üben. Probieren Sie doch einmal mit unserem Rennwagen die verschiedenen Möglichkeiten durch.

10.3 Ein Sprite ändert seine Form

Die Systemroutine CALL PATTERN ermöglicht es, einen Sprite mitten im Programmverlauf in seiner Form zu verändern. Dies ist besonders dann nützlich, wenn ein Sprite in einem selbstprogrammierten Telespiel explodieren soll. Durch eine entsprechende Vertonung (siehe Kapitel 7) kann man solch einen Vorgang sehr wirkungsvoll untermalen.

Definieren wir doch einmal ein Zeichen, das für eine Explosion tauglich ist:

```
* . . *:* . * .    =>    1 0 0 1:1 0 1 0    =>    9 : A
. * . *:* . . *    =>    0 1 0 1:1 0 0 1    =>    5 : 9
. . * .:. . * .    =>    0 0 1 0:0 0 1 0    =>    3 : 2
. . . *:* . * *    =>    0 0 0 1:1 0 1 1    =>    1 : B
* * . *:* . . .    =>    1 1 0 1:1 0 0 0    =>    D : 8
. . * .:. * . .    =>    0 0 1 0:0 1 0 0    =>    3 : 4
. * . *:* . * .    =>    0 1 0 1:1 0 1 0    =>    5 : A
* . . *:* . . *    =>    1 0 0 1:1 0 0 1    =>    9 : 9
```

Sie sehen, daß das obenstehende Muster 'wild' entworfen wurde. Wenn wir unseren Rennwagen mit Hilfe von CALL PATTERN in dieses Zeichen umwandeln wird es aussehen, als ob alle Teile des Wagens kreuz und quer durch die Luft fliegen.

Programmieren wir doch einfach einmal eine kleine Routine, die den Rennwagen explodieren läßt. Über CALL SPRITE wird unser Rennwagen auf die Bildschirmmitte gebracht und dort solange stehen bleiben, bis die Leertaste gedrückt wird, die den Wagen in 'die Luft jagt':

```
10 CALL CLEAR
20 CALL CHAR (128,'99FFBD3C24BDFFBD')
30 CALL CHAR (129,'9A59321BD8345A99')
40 CALL SPRITE (#1,128,7,90,120)
50 CALL KEY (0,K,S)
60 IF S=0 THEN 50
70 IF K<>32 THEN 50
80 CALL PATTERN (#1,129)
90 FOR I=1 TO 50
100 NEXT I
110 GOTO 10
```

Diese Routine wird den ganzen Vorgang so oft wiederholen, bis sie durch Drücken von >FCTN 4< abgebrochen wird.

10.4 Ein Sprite wird größer

Eine sehr nützliche Einrichtung im Zusammenhang mit Sprites ist die CALL MAGNIFY-Routine. Dieser Befehl ermöglicht es, einen Sprite zu vergrößern. Die Vergrößerungsfaktoren, die bei diesem Befehl als Parameter übergeben werden müssen, entnehmen Sie bitte dem Kapitel 8.2.15.

Sicherlich ist es manchmal in Spielen sehr eindrucksvoll, wenn man ein Sprite in doppelter Größe erscheinen lassen kann. Es ist hier sogar möglich mehrere Zeichen (4) zu einem

Sprite zusammen zu fassen (auch hier verweisen wir auf Kapitel 8.2.15).

Wir wollen nun einmal unseren Rennwagen zuerst in normaler Größe auf der Bildschirmmitte darstellen. Um Ihn zu vergrößern drücken Sie bitte irgendeine Taste. Ein weiterer Tastendruck läßt den Sprite wieder in normaler Größe erscheinen.

```
10 CALL CLEAR
20 CALL CHAR (128,'99FFBD3C24BDFFBD')
30 CALL SPRITE (#1,128,7,90,120)
40 CALL KEY (0,K,S)
50 IF S=0 THEN 40
60 CALL MAGNIFY (2)
70 CALL KEY (0,K,S)
80 IF S=0 THEN 70
90 CALL MAGNIFY (1)
100 GOTO 10
```

Anmerkung: Wenn Sie Sprites die aus vier Zeichen bestehen vergrößern (Faktor 4), erscheinen diese in vierfacher Größe.

10.5 Sprites treffen zusammen

Vielleicht haben Sie sich schon einmal Gedanken darüber gemacht wie es realisiert wird, daß in einem Telespiel die verschiedenen Figuren aufeinander reagieren. Eine Antwort darauf wollen wir Ihnen hier geben: Die Systemroutine CALL COINC selektiert das Zusammentreffen zweier oder mehrer Sprites oder einem Sprite und einer bestimmten

Bildschirmposition. Wenn durch CALL COINC ein Zusammentreffen gemeldet wird können Sie entscheiden, was weiter geschehen soll (z.B. Explosion). Mit Hilfe von CALL COINC können Sie auch den Bereich, in dem sich ein Sprite bewegen soll begrenzen bzw. festlegen.

Unsere nächste Routine erzeugt vier Rennwagen und läßt diese wahllos auf dem Bildschirm 'umherfahren'. Es brauchen keine neuen Rennwagen definiert zu werden, da es durch CALL SPRITE möglich ist, mehrere Sprites mit dem gleichen ASCII-Code zu erzeugen. Das Beispielprogramm wird so lange laufen, bis sich zwei Rennwagen treffen und somit das Programm unterbrechen.

```
10 CALL CLEAR
20 CALL CHAR (128,'99FFBD3C24BDFFBD')
30 CALL SPRITE (#1,128,7,90,120)
40 CALL SPRITE (#2,128,2,5,5)
50 CALL SPRITE (#3,128,3,5,220)
60 CALL SPRITE (#4,128,14,150,80)
70 CALL MAGNIFY (2)
80 CALL MOTION (#1,80,90,#2,-50,40)
90 CALL MOTION (#3,100,70,#4,-100,20)
100 CALL COINC (ALL,X)
110 IF NOT X THEN 100
120 STOP
```

Beachten Sie, daß in dem obenstehenden Programm in den Zeilen 80 und 90 mit einem einzigen MOTION-Statement mehrere Sprites angesprochen werden. Die gleiche Möglichkeit haben Sie auch bei allen anderen Befehlen, die der Spritebehandlung dienen.

Wenn Sie mit CALL COINC das Zusammentreffen eines Sprites mit einer Bildschirmposition selektieren wollen, muß im

Parameter ein Toleranzwert mit eingegeben werden. Dieser Wert gibt-in Bildschirmpunkten-den Umkreis um die definierte Position an, innerhalb dessen ein Zusammentreffen gültig sein soll.

10.6 Wo befindet sich ein Sprite

Die meisten von Ihnen kennen sicherlich das Telespiel 'SPACE INVADERS'. Dort wird der Sprite, der dem jeweiligen Spieler zugeordnet ist, fortlaufend durch die Invasoren beschoßen.

Woher wissen nun diese Sprites wo sich der zu bekämpfende Sprite befindet ?

Durch die CALL POSITION - Routine kann man die Position eines oder mehrerer Sprites feststellen. Nun ist es ein leichtes, diese Position auszuwerten und die anderen Sprites danach zu steuern.

In einem Beispielprogramm soll unser Rennwagen schräg über den Bildschirm 'fahren'. Der Wagen wird solange fahren, bis Sie auf irgendeine Taste drücken. Nach dem Tastendruck wird die CALL POSITION - Routine abfragen, ob sich unser Renner im Bereich der Punktzeile 190 und der Punktspalte 256 befindet. Wenn 'JA' wird das Programm unterbrochen:

```
10 CALL CLEAR
20 CALL CHAR (128,'99FFBD3C24BDFFBD')
30 CALL SPRITE (#1,128,7,90,120)
40 CALL MOTION (#1,50,30)
50 CALL KEY (0,K,S)
60 IF S=0 THEN 50
```

```
70 CALL POSITION (#1,PZ,PS)
80 IF PZ>=190 AND PS<=256 THEN STOP
90 GOTO 70
```

10.7 Wie weit befinden sich zwei Sprites voneinander entfernt

Die CALL DISTANCE - Routine gibt die Entfernung zwischen zwei Sprites oder einem Sprite und einer Bildschirmposition an. Hier müssen Sie besonders darauf achten, daß die Entfernung als Quadratzahl angegeben wird.

Anwendung findet dieser Befehl dort, wo man in Spielen eine Explosion programmiert und alle Sprites, die sich in einem bestimmten Umkreis befinden, detonieren lassen will.

In einem weiteren Beispiel mit unserem Rennwagen soll die Beispielroutine unterbrochen werden, wenn der Wagen von der Bildschirmposition 39/80 (Bildschirmzeile 39/Bildschirmspalte 80) 20 Bildpunkte entfernt ist. Um in die Nähe dieses Punktes zu kommen, wird unser Rennwagen fortlaufend schräg ber den Bildschirm fahren.

```
10 CALL CLEAR
20 CALL CHAR (128,'99FFBD3C24BDFFBD')
30 CALL SPRITE (#1,128,7,90,120)
40 CALL MOTION (#1,-30,59)
50 CALL DISTANCE (#1,39,80,X)
60 IF SQR(X)<=20 THEN STOP
70 GOTO 50
```

10.8 Sprites werden gelöscht

Mit der CALL DELSPRITE - Routine ist es möglich Sprites zu löschen. Sie können in dem Parameter spezifizieren,ob einzelne, mehrere oder alle Sprites gelöscht werden sollen. Wenn Sie ein Sprite mit der CALL DELSPRITE-Routine gelöscht haben, steht es für den weiteren Spielverlauf nicht mehr zur Verfügung. Soll der Sprite später noch einmal benötigt werden, müssen Sie ihn mit der CALL SPRITE - Routine neu definieren.

Format:

```
CALL DELSPRITE (#1)
     oder
CALL DELSPRITE (#1,#2,....,#N)
     oder
CALL DELSPRITE (ALL)
```

Unser Rennwagen muß wieder einmal für ein Beispiel herhalten:

In der Bildschirmmitte wird ein Sprite dargestellt, der nach Drücken der Leertaste gelöscht wird. Ein weiteres Drücken dieser Taste läßt den Sprite an seiner Position neu erscheinen.

```
10 CALL CLEAR
20 CALL CHAR (99FFBD3C24BDFFBD')
30 CALL SPRITE (#1,128,7,90,120)
40 CALL KEY (0,K,S)
50 IF S=0 THEN 40
60 CALL DELSPRITE (#1)
70 CALL KEY (0,K,S)
80 IF K=32 THEN 10 ELSE 70
```

10.9 Ein komfortabler Sprite Editor

Im Folgenden wollen wir Ihnen einen Sprite Editor vorstellen, der es Ihnen ermöglicht, auf einfachste Art und Weise Sprites zu erstellen und auszutesten. Sie brauchen sich nicht mehr mit Papier und Bleistift abzumühen, sondern können Ihre Sprites bequem am Bildschirm entwickeln.

Wenn Sie das Programm mit RUN gastartet haben, wird ein Kästchen generiert, in dem ein Cursor erscheint. Der Untergrund dieses Kästchens ist weiß. Diese weiße Farbe signalisiert, daß die einzelnen Punkte (64) aus denen ein Sprite besteht, ausgeschaltet sind. Sie können dann durch Drücken der Taste '1' einen Punkt setzen ('einschalten'). Die Farbe unter dem Coursor wird dann schwarz sein. Mit den Cursor-Tasten (Pfeiltasten E,X,S,D) können Sie den Cursor frei in dem Kästchen bewegen. Wo immer Sie wollen können Sie durch Drücken von '1' einen Punkt einschalten. Sollten Sie während diesem Vorgang einen Punkt, den Sie gesetzt haben, wieder ausschalten wollen, können Sie dies mit der Taste '0' tun. Nun wird die Farbe unter dem Cursor wieder weiß werden.

Wenn Sie Ihren Sprite fertig entworfen haben, können Sie ihn durch betätigen der Taste 'P' auf den Bildschirm holen. Ist Ihr Sprite endlich entworfen und auf dem Bildschirm ausgegeben, haben Sie folgende Optionen:

[M] - Vergrößerung des Sprites.

[C] - Farbe des Sprites verändern.

[B] - Verändern der Hintergrundfarbe des Sprites.

[X,E,S,D] - Mit diesen Taste können Sie den neu entwickelten Sprite in Bewegung versetzen.

Sollte Ihnen Ihr Sprite nicht zusagen, können Sie durch Drücken der Taste 'T' die Form nocheinmal verändern. Sollten die gewünschten Änderungen sehr umfangreich sein, empfiehlt es sich, mit der Taste 'A' ein neues Sprite zu definieren.

Wenn Sie die Basic - Statements, die zum Erzeugen Ihres Sprites nötig sind, auf dem Bildschirm ausgeben wollen, drücken Sie einfach die Taste 'L'.

Nach dem Programmlisting folgt eine ausführliche Beschreibung des Editors, damit Sie die Arbeitsweise des Programmes verstehen.

```
10 REM ** SPRITE EDITOR **
20 DIM L(16,16)
25 O=1
30 O1=7
40 CALL CHAR (100,' ')
50 CALL CHAR (101,'FFFFFFFFFFFFFFFF')
60 CALL CHAR (102,'FFFFC3C3C3C3FFFF')
70 CALL COLOR (9,2,16)
80 CALL CLEAR
90 DISPLAY AT(1,10):'SPRITE EDITOR'
100 FOR I=1 TO 16
104 CALL HCHAR (4+I,2,100,16)
108 NEXT I
110 CALL MAGNIFY (1)
120 IF K=84 THEN 140
130 CALL SCREEN (8)
140 CALL DELSPRITE (ALL)
150 CALL SPRITE (#28,102,14,32,8)
160 CALL HCHAR (21,1,32,31)
165 CALL HCHAR (22,2,32,31)
170 DISPLAY AT(22,2):'E=UP-X=DOWN-S=LEFT-D=RIGHT'
180 DISPLAY AT(23,2):'1=PUNKT EIN - 0=PUNKT AUS'
190 DISPLAY AT(24,2):'P=SPRITE AUSGEBN'
200 N=1
205 O2=1
210 CALL KEY (0,K,S)
220 IF S=0 THEN 210
230 IF K=48 THEN ZEI=100
240 IF K=49 THEN ZEI=101
250 IF K=83 THEN O2=O2-1 :: GOTO 300
260 IF K=68 THEN O2=O2+1 :: GOTO 300
270 IF K=69 THEN N=N-1 :: GOTO 300
280 IF K=88 THEN N=N+1 :: GOTO 300
290 IF K=80 THEN 380
300 IF O2<1 THEN O2=16
```

```
310 IF O2>16 THEN O2=1
320 IF N<1 THEN N=16
330 IF N>16 THEN N=1
340 CALL LOCATE (#28,(8*N)+25,8*O2+1)
350 CALL HCHAR (4+N,1+O2,ZEI)
360 CALL SOUND (20,200,5)
370 GOTO 210
380 CALL DELSPRITE (ALL)
390 CALL HCHAR (21,1,32,128)
400 DISPLAY AT(22,2):'BITTE WARTEN !!!!'
410 FOR N=1 TO 16
420 FOR O2=1 TO 16
430 CALL GCHAR (4+N,1+O2,SS)
440 SS=SS-100
450 L(N,O2)=SS
460 NEXT O2
470 NEXT N
480 HD$='0123456789ABCDEF'
490 F$=' '
500 FOR N=1 TO 16
510 LB=L(N,5)*8+L(N,6)*4+L(N,7)*2+L(N,8)+1
520 HB=L(N,1)*8+L(N,2)*4+L(N,3)*2+L(N,4)+1
530 F$=F$&SEG$(HD$,HB,1)&SEG$(HD$,LB,1)
540 NEXT N
550 FOR N=1 TO 16
560 LB=L(N,13)*8+L(N,14)*4+L(N,15)*2+L(N,16)+1
570 HB=L(N,9)*8+L(N,10)*4+L(N,11)*2+L(N,12)+1
580 F$=F$&SEG$(HD$,HB,1)&SEG$(HD$,LB,1)
590 NEXT N
600 CALL CHAR (104,F$)
610 CALL MAGNIFY (3)
620 FF=3
625 F=4
630 CALL SPRITE (#1,104,C1,50,170,0,0)
640 DISPLAY AT(21,2):'C=FARBE-M=VERGR.-T=AENDERN'
```

```
650 DISPLAY AT(22,2):'A=NEU-Q=ENDE-B=HINTERGRUND'
660 DISPLAY AT(23,2):'E=UP-X=DOWN-S=LEFT-D=RIGHT'
670 DISPLAY AT(24,8):'L=BASIC STATEMENTS'
680 CALL KEY (0,K,S)
690 IF K=76 THEN 900
700 IF K=81 THEN 890
710 IF K=65 THEN GOSUB 1180
720 IF K=84 THEN 110
730 IF K=77 THEN 940
740 IF K=67 THEN 1150
750 IF K=83 THEN J=J-2
760 IF K=68 THEN J=J+2
770 IF K=69 THEN K1=K1-2
780 IF K=88 THEN K1=K1+2
785 IF K1>120 THEN K1=120
790 IF K1<-200 THEN K1=-200
800 IF J>120 THEN J=120
810 IF J<-200 THEN J=-200
820 CALL MOTION (#1,K1,J)
830 GOTO 680
840 CALL MAGNIFY (F)
850 FF=F
860 IF F=3 THEN F=4 ELSE F=3
870 FOR I=1 TO 20
875 NEXT I
880 GOTO 680
890 STOP
900 REM
910 CALL CLEAR
920 PRINT '** PROGRAMMLISTING **'
930 CALL DELSPRITE (ALL)
940 PRINT
950 PRINT '=>10 CALL CHAR (104,X';
960 FOR I=1 TO 64 :: PRINT SEG$(F$,I,1); :: NEXT I
970 PRINT 'X)'
```

```
980 PRINT '=>20 CALL SCREEN (';O;')'
990 PRINT '=>30 CALL MAGNIFY (';FF;')'
1000 PRINT '=>40 CALL SPRITE (#1,104,';O1;',150,150,';K1;','
        ;J;')'
1010 PRINT '=>50 CALL KEY (0,K,S)'
1020 PRINT '=>60 IF K=68 THEN J=J+2'
1030 PRINT '=>70 IF K=83 THEN J=J-2'
1040 PRINT '=>80 IF K=88 THEN K1=K1+2'
1050 PRINT '=>90 IF K=69 THEN K1=K1-2'
1060 PRINT '=>100 CALL MOTION (#1,K1,J)'
1070 PRINT '=>110 GOTO 50'
1080 FOR I=1 TO 5 :: PRINT :: NEXT I
1090 DISPLAY AT(21,3):'A=NEU-Q=ENDE'
1100 CALL KEY (0,K,S)
1110 IF S=0 THEN 1100
1120 IF K=81 THEN 890
1130 IF K=65 THEN 10
1140 GOTO 1100
1150 O1=O1+1
1155 IF O1>16 THEN 1170
1160 CALL COLOR (#1,O1) :: GOTO 680
1170 O1=2 :: CALL COLOR (#1,O1) :: GOTO 680
1180 O=O+1 :: IF O=17 THEN O=2
1190 CALL SCREEN (O)
1200 RETURN
```

Programmbeschreibung:

Zeilennummer	Erläuterungen
20	Dimensionierung der benötigten Datenfelder.
25 - 30	Hier werden die Variablen O und O1 auf Ihre Anfangswerte gesetzt.
40 - 60	Die Graphikzeichen für die Bildschirmdarstellung werden definiert.
70	Der Bildschirm erhält seine Farbe.
80 - 210	Die Bildschirmgraphik wird generiert. In Zeile 150 wird Sprite #28 definiert, der als Cursor dient.
220 - 330	Abfrageroutine, die feststellt welche Steuertaste gedrückt wurde.
340	Rücksetzen des Cursors an seine ursprüngliche Position.
380 - 610	Der entworfene Sprite wird übernommen und mit CALL CHAR in Zeile 600 definiert. In den Zeilen 510 - 580 wird der Mus-

Zeilennummer	Erläuterungen
	sterstring erstellt.
610 - 680	Der entworfene Sprite wird auf dem Bilschirm dargestellt und kann über die angegebenen Optionen manipuliert werden.
690 - 890	Abfrageroutine für die Steuertasten (690-810) und Steuerung der angwählten Optionen (820-890).
900 - 1140	Hier wird auf dem Bildschirm das BASIC - Listing angezeigt, das eingetippt werden muß, um den entworfenen Sprite in anderen Programmen zu definieren.
1180-1200	Unterprogramm für die Farbe des Bildschirmes.

ANHANG

Um die Arbeit mit diesem Buch zu optimieren, wollen wir Ihnen hier alle wichtigen Tabellen abdrucken. Eine Reihe von Arbeitsblättern im Anschluß soll Ihnen den Entwurf von Graphikzeichen und Sprites erleichertern.

I. Die ASCII-CODES

32	Leerstaste (Space)	81	Q
33	! (Ausrufezeichen)	82	R
34	" (Anführungszeichen)	83	S
35	# (Nummernzeichen)	84	T
36	$ (Dollarzeichen)	85	U
37	% (Prozentzeichen)	86	V
38	& (kaufm. Und)	87	W
39	' (Apostroph)	88	X
40	((offene Klammer)	89	Y
41	) (geschlossene Klammer)	90	Z
42	* (Asterisk)	91	[(li.eckige Kl.)
43	+ (Pluszeichen)	92	\ (reverse Slash)
44	, (Komma)	93	] (re.eckige Kl.)
45	- (Minuszeichen)	94	↑ (Exponentation)
46	. (Punkt)	95	_ (Linie)
47	/ (Slash)	96	` (Grave)
48	0	97	a
49	1	98	b
50	2	99	c
51	3	100	d
52	4	101	e
53	5	102	f

54	6	103	g
55	7	104	h
56	8	105	i
57	9	106	j
58	: (Doppelpunkt)	107	k
59	; (Semikolon)	108	l
60	< (Kleinerzeichen)	109	m
61	= (Gleichheitszeichen)	110	n
62	> (Größerzeichen)	111	o
63	? (Fragezeichen)	112	p
64	@ (At-Zeichen)	113	q
65	A	114	r
66	B	115	s
67	C	116	t
68	D	117	u
69	E	118	v
70	F	119	w
71	G	120	x
72	H	121	y
73	I	122	z
74	J	123	{ (re.geschw.Klammer)
75	K	124	¦ (Beistrich)
76	L	125	} (li.geschw.Klammer)
77	M	126	~ (Tilde)
78	N	127	DEL
79	O		
80	P		

Anmerkung: Einige Zeichen des vorausgegangenen ASCII-Codes konnten aus techn. Gründen nicht so abgedruckt werden, wie sie beim TI tatsächlich vorhanden sind. Son sind z.B. alle Kleinbuchstaben beim TI als kleine Großbuchstaben im Zeichensatz.

II. Zeichengruppen

TI-BASIC	EXTENDED BASIC	ASCII-CODES
	0	30 - 31
1	1	32 - 39
2	2	40 - 47
3	3	48 - 55
4	4	56 - 63
5	5	64 - 71
6	6	72 - 79
7	7	80 - 87
8	8	88 - 95
9	9	96 -103
10	10	104 -111
11	11	112 -119
12	12	120 -127
13	13	128 -135
14	14	136 -143
15		144 -151
16		152 -159

Diese Zeichengruppen benötigen Sie in Verbindung mit dem CALL COLOR - Befehl.

III. Tasteneinheiten

Einheit	Tastatur
0	Konsole
1	linke Seite der Konsole
2	rechte Seite der Konsole
3	nicht definiert
4	nicht definiert
5	nicht definiert

IV. Umrechnungstabelle

Dezimal	Hexadezimal	Binär
0	0	0 0 0 0
1	1	0 0 0 1
2	2	0 0 1 0
3	3	0 0 1 1
4	4	0 1 0 0
5	5	0 1 0 1
6	6	0 1 1 0
7	7	0 1 1 1
8	8	1 0 0 0
9	9	1 0 0 1
10	A	1 0 1 0
11	B	1 0 1 1
12	C	1 1 0 0
13	D	1 1 0 1
14	E	1 1 1 0
15	F	1 1 1 1

V. Farbcode-Tabelle

1 => Transparent
2 => Schwarz
3 => Mittelgrün
4 => Hellgrün
5 => Dunkelblau
6 => Hellblau
7 => Dunkelrot
8 => Kornblumenblau
9 => Mittelrot
10 => Hellrot
11 => Dunkelgelb
12 => Hellgelb
13 => Dunkelgrün
14 => Magenta
15 => Grau
16 => Weiß

VI. Zeichencodes für die geteilte Tastatur

CODE	TASTEN
0	X,M
1	A,H
2	S,J
3	D,K
4	W,U
5	E,I
6	R,O
7	2,7
8	3,8
9	4,9
10	5,0
11	T,P
12	F,L
13	V,. (Punkt)
14	C,, (Komma)
15	Z,N
16	B,/ (Slash)
17	G,; (Semikolon)
18	Q,Y
19	1,6

Die geteilte Tastatur können Sie in Verbindung mit der Routine CALL KEY verwenden, um die Eingabe von bestimmten Tasten oder Tastengruppen zu kontrollieren. Siehe hier auch unter III. (Tastencodes) nach.

VII. Tabelle der Noten

FREQUENZ	NOTE	:	FREQUENZ	NOTE
110	A	:	440	A
116	A#	:	466	A#
123	H	:	493	H
130	C	:	523	C
138	C#	:	554	C#
146	D	:	587	D
155	D#	:	622	D#
164	E	:	659	E
174	F	:	698	F
184	F#	:	739	F#
195	G	:	783	G
207	G#	:	830	G#
220	A	:	880	A
233	A#	:	932	A#
246	H	:	987	H
261	C	:	1046	C
277	C#	:	1108	C#
293	D	:	1174	D
311	D#	:	1244	D#
329	E	:	1318	E
349	F	:	1396	F
369	F#	:	1479	F#
391	G	:	1567	G
415	G#	:	1661	G#

Diese Tonfrequenzen werden in Verbindung mit dem CALL SOUND-Befehl benütigt.

VIII. Liste der Fehlermeldungen

Die folgende Liste ist nach den Fehlernummern sortiert, die bei Verwendung des Moduls EXTENDED BASIC, beim Unterprogramm CALL ERR auftreten können.

Fehlernummer	Fehler
10	NUMERIC OVERFLOW
14	SYNTAX ERROR
16	ILLEGAL AFTER SUBPROGRAM
17	UNMATCHED QUOTES
19	NAME TOO LONG
20	UNRECOGNIZED CHARACTER
24	STRING-NUMBER MISMATCH
25	OPTION BASE ERROR
28	IMPROPERLY USED NAME
36	IMAGE ERROR
39	MEMORY FULL
40	STACK OVERFLOW
43	NEXT WITHOUT FOR
44	FOR-NEXT NESTING
47	MUST BE IN SUBPROGRAM
48	RECURSIVE SUBPROGRAM CALL
49	MISSING SUBEND
51	RETURN WITHOUT GOSUB
54	STRING TRUNCATED
56	SPEECH STRING TOO LONG
57	BAD SUBSCRIPT
60	LINE NOT FOUND
61	BAD LINE NUMBER
62	LINE TOO LONG
67	CAN'T CONTINUE

69	COMMAND ILLEGAL IN PROGRAM
70	ONLY LEGAL IN A PROGRAM
74	BAD ARGUMENT
78	NO PROGRAM PRESENT
79	BAD VALUE
81	INCORRECT ARGUMENT LIST
83	INPUT ERROR
84	DATA ERROR
97	PROTECTION VIOLATION
109	FILE ERROR
130	I/O ERROR
135	SUBPROGRAMM NOT FOUND

A C H T U N G

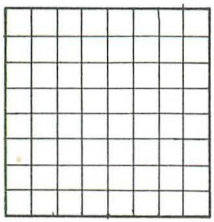
Diese Fehlertabelle ist für TI-BASIC n i c h t gültig!!!

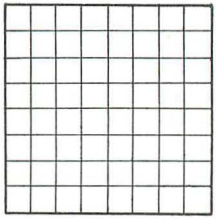
IX. Arbeitsblätter zum Entwurf von Sprites

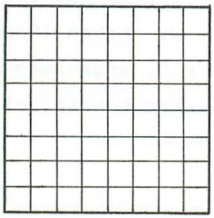
Die folgenden Seiten sollen Ihnen beim Entwurf von Sprites oder Graphikzeichen gute Dienste leisten. Wir haben Ihnen hier das Zeichnen der Entwurfskästchen (8 x 8) abgenommen. Jedes Blatt ist für den Entwurf von 6 Zeichen konzipiert.

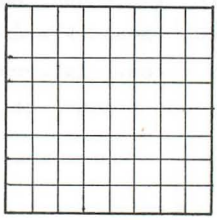
Entwurfsblatt für Sprites

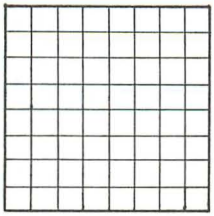
****

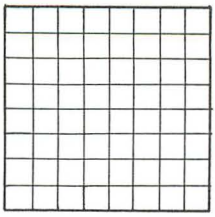






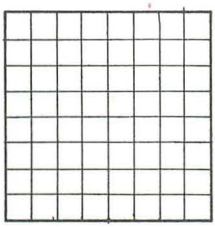


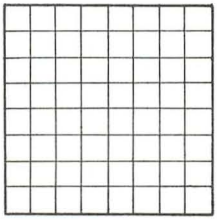


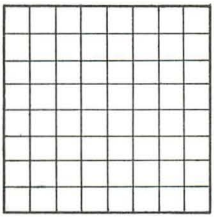


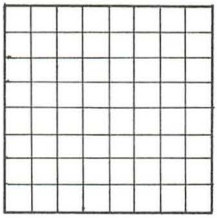
Entwurfsblatt für Sprites

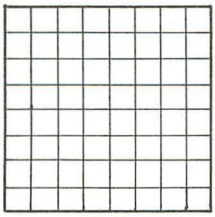
****

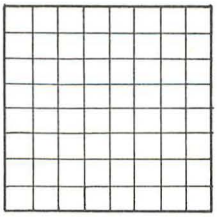






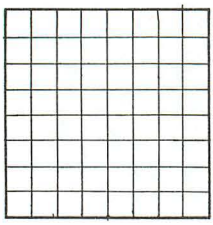


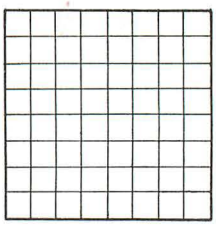


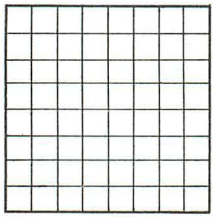


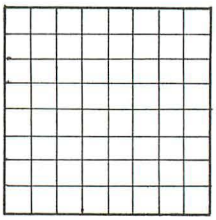
Entwurfsblatt für Sprites

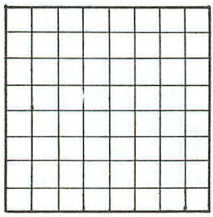
****

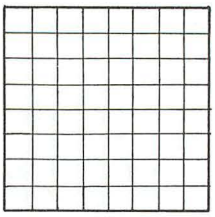






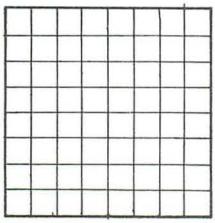


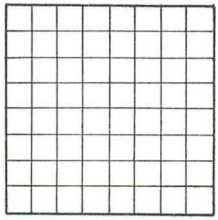


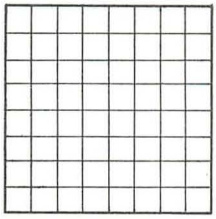


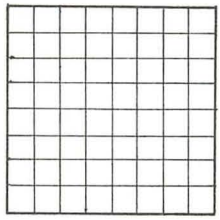
Entwurfsblatt für Sprites

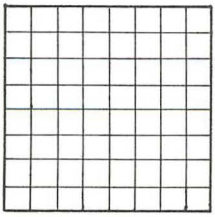
****

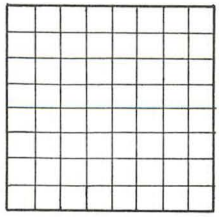






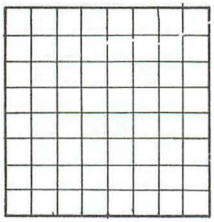


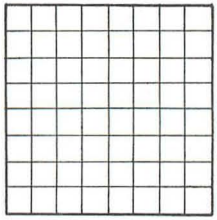


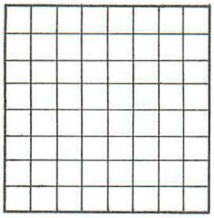


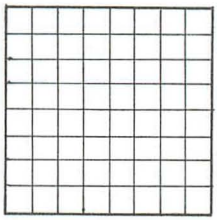
Entwurfsblatt für Sprites

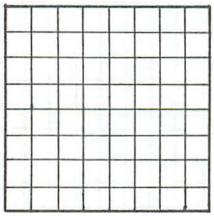
****

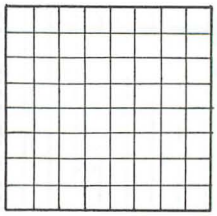










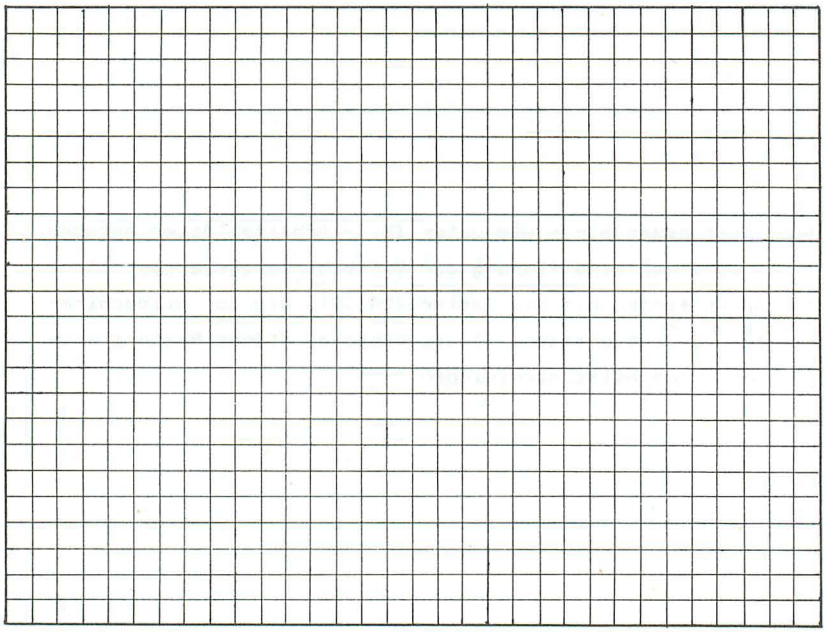


X. Arbeitsblätter zum Entwurf einer Bildschirmausgabe

Auch hier haben wir - wie unter IX. - Arbeitsblätter entworfen, die Ihnen eine Planung der Bildschirmausgabe gestatten. Pro Blatt finden Sie ein Raster (24x32), die der Bildschirmaufteilung entsprechen. Jedes Kästchen dieses Rasters entspricht einem Bildschirmzeichen.

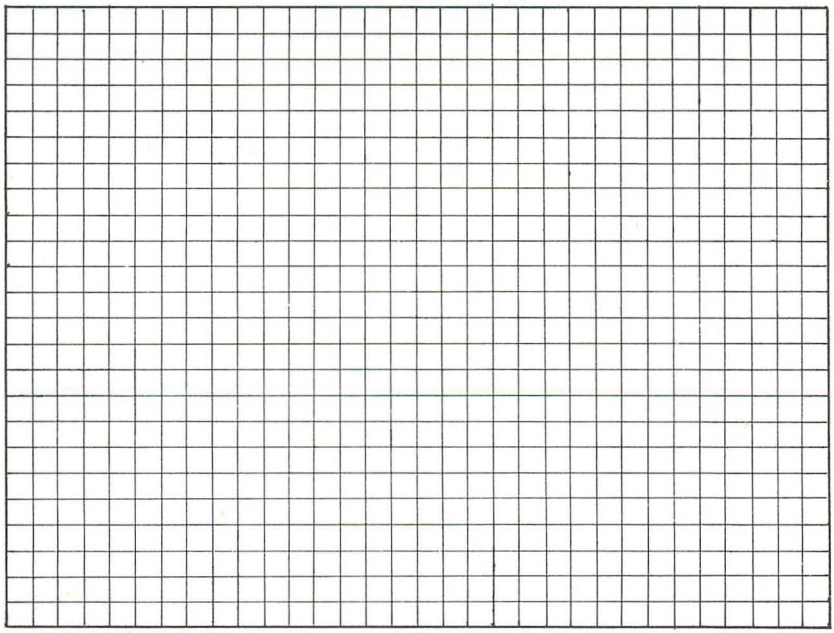
Entwurfsblatt für die Bildschirmausgabe

****



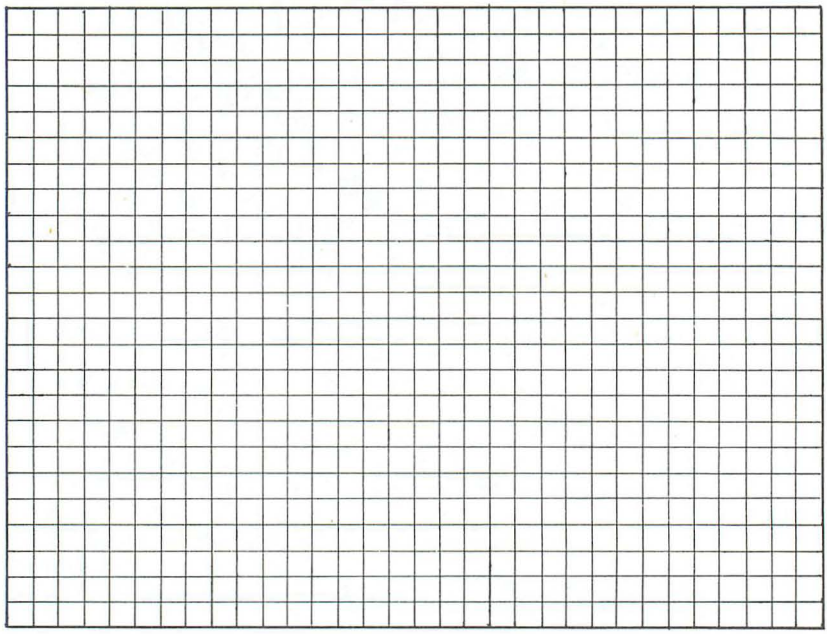
Entwurfsblatt für die Bildschirmausgabe

****



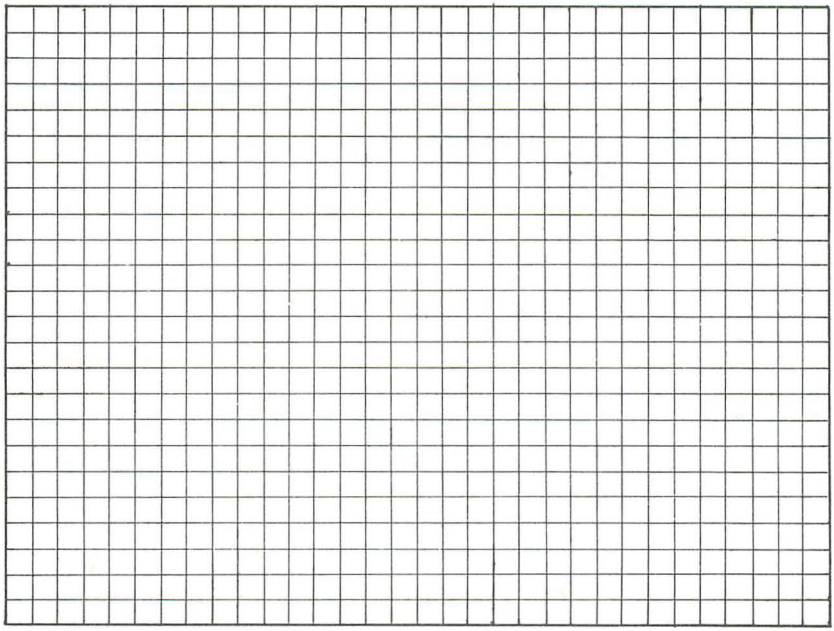
Entwurfsblatt für die Bildschirmausgabe

****



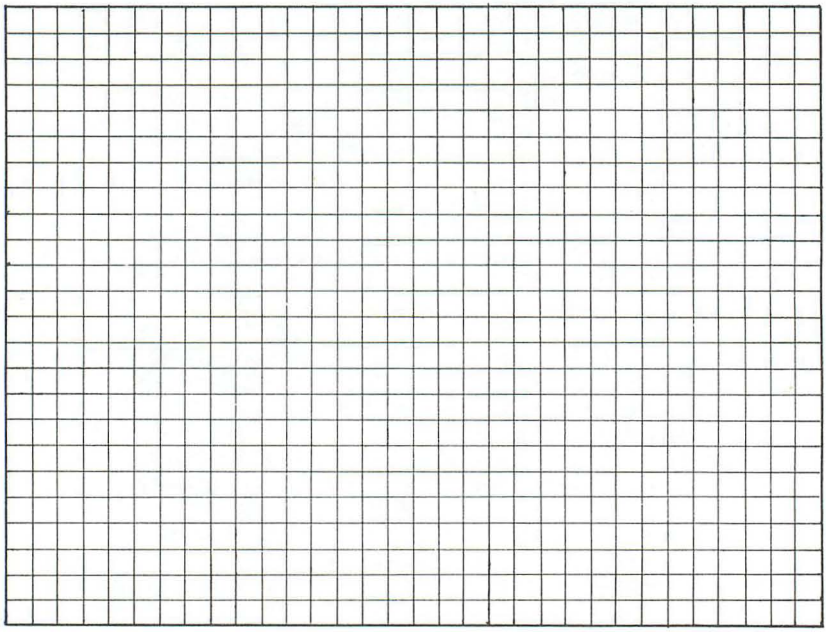
Entwurfsblatt für die Bildschirmausgabe

****



Entwurfsblatt für die Bildschirmausgabe

****



MITMACHEN!

Anfang 1984 erscheint DATA BECKER's GROSSE TI-99 PROGRAMMSAMMLUNG. Ein neues Superbuch zum TI-99 mit über 300 Seiten. Voll mit interessanten Programmen zum TI-99. Vom Spielehit bis zur BASIC-Erweiterung. Alle Programme getestet und fertig zum Eintippen. Geschrieben von Programmierern aus ganz Deutschland.

An diesem neuen Superbuch können Sie mitschreiben. Wie? - Ganz einfach:

Sie senden uns Ihr Programm auf Datenträger (Cassette bzw. Diskette) und als Listing mit entsprechenden Bedienungshinweisen. Voraussetzung ist, daß Sie das alleinige Copyright für Ihr Programm besitzen und Ihr Programm noch nicht veröffentlicht wurde. Selbstverständlich können Sie uns auch mehrere Programme senden.

Wir testen Ihr Programm und teilen Ihnen innerhalb von 4 Wochen mit, ob wir es übernehmen möchten. In diesem Falle machen wir Ihnen ein Angebot über den Kauf Ihres Programmes. Wir zahlen je nach Art und Umfang pro Programm zwischen DM 100,-- und DM 400,--, nehmen Sie mit Kurzbiographie in das Autorenverzeichnis auf und schicken Ihnen sofort nach Erscheinen des Buches zwei Freiexemplare.

Sie wollen mitmachen? - Dann senden Sie bitte Ihr Programm an:

DATA BECKER
Aktion "MITMACHEN"
Merowingerstr. 30
4000 Düsseldorf